

Spring Security UI Plugin

Reference Documentation



GRAILS

Spring Security UI Plugin - Reference Documentation

Burt Beckwith

Version unspecified

Table of Contents

1. Introduction to the Spring Security UI Plugin.....	1
1.1. Installation.....	1
1.2. Release History	1
2. User Management	3
2.1. User search	3
2.2. User edit	4
2.3. User creation	6
3. Role Management	7
3.1. Role search.....	7
3.2. Role edit	7
3.3. Role creation	8
4. Requestmap Management	9
4.1. Requestmap search	9
4.2. Requestmap edit	9
4.3. Requestmap creation	10
5. User Registration	11
5.1. Registration	11
5.2. Configuration	13
6. E-Mail Validation Parameter.....	14
6.1. Mail configuration	14
6.2. Notes	14
6.3. RegistrationCode search.....	15
6.4. RegistrationCode edit	16
7. Forgot Password.....	17
7.1. Forgot Password.....	17
7.2. Configuration	19
7.3. Email Validation Configuration	19
7.4. Challenge Questions Configuration	19
7.5. Mail configuration	21
7.6. Notes	21
8. ACL Management.....	22
8.1. AclClass Management	22
8.2. AclSid Management.....	23
8.3. AclObjectIdentity Management	25
8.4. AclEntry Management	27
9. Persistent Cookie Management	30
9.1. Persistent logins search	30
9.2. Persistent logins edit	31

9.3. Persistent logins creation	31
10. Security Configuration UI	32
10.1. Security Configuration	32
10.2. Mappings	32
10.3. Current Authentication	33
10.4. User Cache	33
10.5. Filter Chains	34
10.6. Logout Handlers	35
10.7. Voters	35
10.8. Authentication Providers	35
10.9. Secure Channel Definition	36
11. Customization	37
11.1. s2ui-override script	37
11.2. I18N	39
11.3. application.groovy attributes	39
11.4. CSS and JavaScript	40
11.5. Password Hashing	42
11.6. Password Verification	44
12. Scripts	45
12.1. s2ui-override	45
12.2. s2ui-create-challenge-questions	45

Chapter 1. Introduction to the Spring Security UI Plugin

The Spring Security UI plugin provides CRUD screens and other user management workflows.

The CRUD screens are protected from cross-site request forgery (CSRF) attacks through the use of the `useToken` attribute in forms. More information is available at <http://docs.grails.org/latest/guide/theWebLayer.html#formtokens>.

Non-default functionality is available only if the feature is available; this includes the ACL controllers and views which are enabled if the `ACL plugin` is installed, Requestmaps support which is available if `grails.plugin.springsecurity.securityConfigType` is set to `"Requestmap"` or `SecurityConfigType.Requestmap` in `application.groovy`, and persistent cookies support which is enabled if it has been configured with the `s2-create-persistent-token` script.

1.1. Installation

Add an entry in the `dependencies` block of your `build.gradle` file, changing the version as needed:

`build.gradle`

```
dependencies {  
    ...  
    compile 'org.grails.plugins:spring-security-ui:4.0.0.BUILD-SNAPSHOT'  
    ...  
}
```



Master branch is for Grails 4, which uses Spring Security 5.1.2 and Spring 5 See the `spring-security-core` plugin for details such as how password encodings have changed (salts).



Version 3.1.x is only compatible with Grails 3.3.x or higher.

For previous Grails 3 versions (3.0.x, 3.1.x and 3.2.x) use:

`build.gradle`

```
dependencies {  
    ...  
    compile 'org.grails.plugins:spring-security-ui:3.0.2'  
    ...  
}
```

Also be sure to update the versions when new releases are available.

1.2. Release History

- Feb 14, 2018

- 3.1.2 release
- Sep 27, 2017
 - 3.1.1 release
- Sep 26, 2017
 - 3.1.0 release
- Jul 28, 2017
 - 3.0.2 release
- Jul 27, 2017
 - 3.0.1 release
- April 15, 2016
 - 3.0.0.M2 release
- December 21, 2015
 - 3.0.0.M1 release
- December 21, 2015
 - 1.0-RC3 release
- May 20, 2014
 - 1.0-RC2 release
- November 11, 2013
 - 1.0-RC1 release
 - [JIRA Issues](#)
- February 12, 2012
 - 0.2 release
 - [JIRA Issues](#)
- September 14, 2010
 - 0.1.2 release
 - [JIRA Issues](#)
- July 27, 2010
 - 0.1.1 release
- July 26, 2010
 - initial 0.1 release

Chapter 2. User Management

2.1. User search

The default action for the User controller is search. By default only the standard fields (`username`, `enabled`, `accountExpired`, `accountLocked`, and `passwordExpired`) are available but this is customizable with the `s2ui-override` script - see the [Customization](#) section for details.

You can search by any combination of fields, and the `username` field has an Ajax autocomplete to assist in finding instances. In this screenshot you can see that an `email` field has been added to the domain class and UI. Leave all fields empty and all checkboxes set at "Either" to return all instances.

The screenshot shows the Spring Security Management Console interface. At the top, there is a navigation bar with links: Users, Roles, Registration Code, and Security Info. The main title is "Spring Security Management Console" and it indicates the user is logged in as "admin" with a "Logout" link. Below this is a "User Search" form. The form has two input fields for "Username:" and "Email:". Below these are four rows of checkboxes for "Enabled:", "Account Expired:", "Account Locked:", and "Password Expired:". Each row has three columns: "True", "False", and "Either". The "Either" column for all four rows has a checked checkbox. A "Search" button is located at the bottom left of the form.

	True	False	Either
Enabled:	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Account Expired:	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Account Locked:	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Password Expired:	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

This example shows a search for usernames containing 'adm' (the search is case-insensitive and the search string can appear anywhere in the username). Results are shown paginated in groups of 10. All of the column headers are clickable and will sort the results by that field.

User Search

Username:

Email:

	True	False	Either
Enabled:	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Account Expired:	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Account Locked:	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Password Expired:	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Username	Email	Enabled	Account Expired	Account Locked	Password Expired
farzadmirezai	farzadmirezai@foo.com	True	False	False	False
cadmus	cadmus@foo.com	True	False	False	False
chenreadme	chenreadme@foo.com	True	False	False	False
admatha	admatha@foo.com	True	False	False	False
sajjadmohebi	sajjadmohebi@foo.com	True	False	False	False
cadmo	cadmo@foo.com	True	False	False	False
padmawar	padmawar@foo.com	True	False	False	False
padma	padma@foo.com	True	False	False	False
ysunadmin	ysunadmin@foo.com	True	False	False	False
farzadmokh	farzadmokh@foo.com	True	False	False	False

1 [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [Next](#)

Showing 1 through 10 out of 100.

2.2. User edit

After clicking through to the 'admin' User you get to the edit page (there are no view pages):

Edit User

User Details

Roles

Username

admin

Email

admin@foo.com

Password

.....

Enabled

☒

Account Expired

☐

Account Locked

☐

Password Expired

☐

Update

Delete

Login as user

You can update any of the attributes or delete the User. You can see that there's a "Login as user" button here - that is only shown if you're authenticated with a User who is granted `ROLE_SWITCH_USER` (this role name can be configured in `application.groovy`):

This allows you to temporarily assume the identity of another User (see [the Spring Security Core plugin documentation](#) for more information about switch-user). The "Logged in as ..." information in the top right of the screen will change to show that you're running as another User and provide a link to switch back. The role name `ROLE_SWITCH_USER` is the default but you can change the value with the `grails.plugin.springsecurity.ui.switchUserRoleName` setting in `application.groovy`.

If you click the Roles tab you can see the roles granted to this User and can click through to its edit page:

Edit User

User Details

Roles

☒ `ROLE_ADMIN`

☒ `ROLE_SWITCH_USER`

☐ `ROLE_USER`

Update

Delete

2.3. User creation

You can create new Users by going to </user/create> or by clicking the **Create** action in the **Users** menu.

[Users](#) [Roles](#) [Registration Code](#) [Security Info](#)

Spring Security Management Console

Logged in as admin ([Logout](#))

Create User

User Details

Roles

Username

Email

Password

Enabled

☐

Account Expired

☐

Account Locked

☐

Password Expired

☐

Create

Chapter 3. Role Management

3.1. Role search

The default action for the Role controller is search. By default only the **authority** field is available but this is customizable with the **s2ui-override** script - see the **Customization** section for details.

The **authority** field has an Ajax autocomplete to assist in finding instances. Leave the field empty to return all instances.



The screenshot shows the 'Spring Security Management Console' interface. At the top, there is a navigation bar with links: 'Users', 'Roles', 'Registration Code', and 'Security Info'. The user is logged in as 'admin' with a 'Logout' link. The main heading is 'Spring Security Management Console'. Below this, there is a 'Role Search' section. It contains a text input field labeled 'Authority:' and a 'Search' button.

Search is case-insensitive and the search string can appear anywhere in the name (and you can omit the **ROLE_** prefix). Results are shown paginated in groups of 10 but if there's only one result you'll be forwarded to the edit page for that Role. The **authority** column header is clickable and will sort the results by that field.

3.2. Role edit

After clicking through to a Role you get to the edit page (there are no view pages):



The screenshot shows the 'Spring Security Management Console' interface for editing a role. The navigation bar is the same as in the previous screenshot. The main heading is 'Spring Security Management Console'. Below this, there is a section titled 'Edit Role'. It contains two tabs: 'Role Details' (selected) and 'Users'. The 'Role Details' tab shows a text input field labeled 'Authority' with the value 'ROLE_ADMIN'. Below the input field, there are two buttons: 'Update' and 'Delete'.

You can update any of the attributes or delete the Role. Any user that had been granted the Role will lose the grant but otherwise be unaffected.

If you click the Users tab you can see which users have a grant for this Role and can click through to

their edit page:

[Users](#) [Roles](#) [Registration Code](#) [Security Info](#)

Spring Security Management Console

Logged in as admin ([Logout](#))

Edit Role

 Role Details

 Users

admin

Update

Delete

3.3. Role creation

You can create new Roles by going to [/role/create](#) or by clicking the **Create** action in the **Roles** menu.

[Users](#) [Roles](#) [Registration Code](#) [Security Info](#)

Spring Security Management Console

Logged in as admin ([Logout](#))

Create Role

Authority

Create

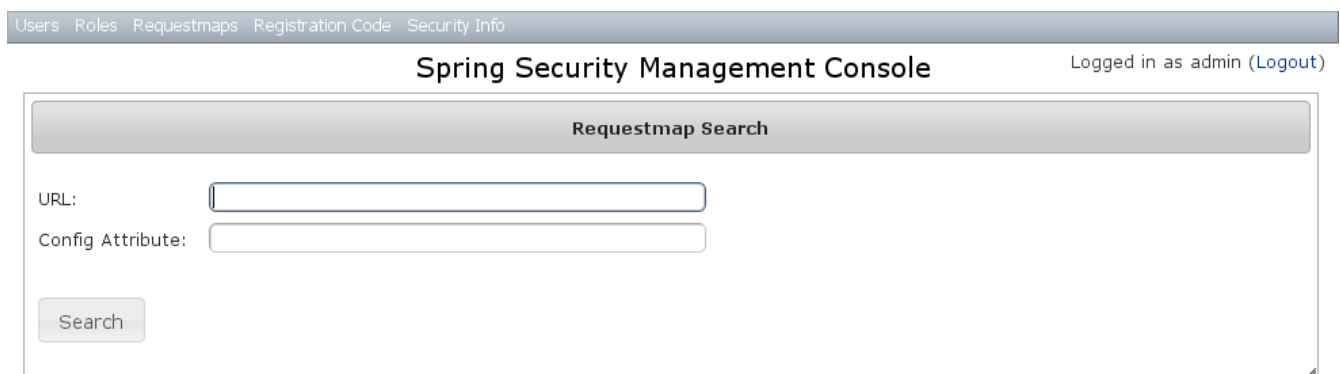
Chapter 4. Requestmap Management

The default approach to securing URLs is with annotations, so the Requestmaps menu is only shown if `grails.plugin.springsecurity.securityConfigType` has the value `"Requestmap"` or `SecurityConfigType.Requestmap` in `application.groovy`.

4.1. Requestmap search

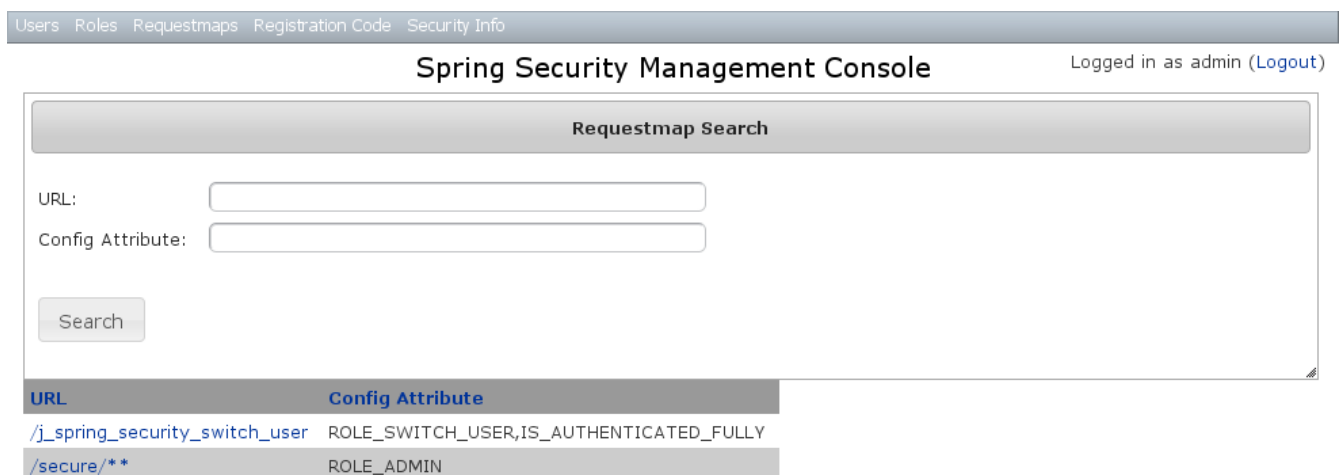
The default action for the Requestmap controller is search. By default only the standard fields (`url` and `configAttribute`) are available but this is customizable with the `s2ui-override` script - see the [Customization](#) section for details.

You can search by any combination of fields, and the `url` and `configAttribute` fields have an Ajax autocomplete to assist in finding instances. Leave both fields empty to return all instances.



The screenshot shows the Spring Security Management Console interface. At the top, there is a navigation bar with links: Users, Roles, Requestmaps, Registration Code, and Security Info. The title "Spring Security Management Console" is centered, and on the right, it says "Logged in as admin (Logout)". Below this is a section titled "Requestmap Search". It contains two input fields: "URL:" and "Config Attribute:". Below these fields is a "Search" button.

Searching is case-insensitive and the search string can appear anywhere in the field. Results are shown paginated in groups of 10 and you can click on either header to sort by that field:



The screenshot shows the same Spring Security Management Console interface, but now displaying search results. The "Requestmap Search" section is still at the top. Below it, there is a table with two columns: "URL" and "Config Attribute". The table contains two rows of results:

URL	Config Attribute
/j_spring_security_switch_user	ROLE_SWITCH_USER,IS_AUTHENTICATED_FULLY
/secure/**	ROLE_ADMIN

Showing 1 through 2 out of 2.

4.2. Requestmap edit

After clicking through to a Requestmap you get to the edit page (there are no view pages):

Edit Requestmap

URL

Config Attribute

Update

Delete

You can update any of the attributes or delete the Requestmap. Editing or deleting a Requestmap resets the cache of loaded instances, so your changes will take effect immediately.

4.3. Requestmap creation

You can create new Requestmaps by going to </requestmap/create> or by clicking the **Create** action in the **Requestmaps** menu.

Create Requestmap

URL

Config Attribute

Create

Creating a Requestmap resets the cache of loaded instances, so your changes will take effect immediately.

Chapter 5. User Registration

Most of the plugin's controllers are intended to be part of a backend admin application, but the Registration and Forgot Password workflows are expected to be user-facing. So they're not available in the admin menu like the User, Role, and other backend functionality - you'll need to expose them to your users.

One way to do this is to replace the default `login.gsp` that's provided by the Spring Security Core plugin with this plugin's version. You can do this by running `grails s2ui-override auth - s2ui-override` script - see the [Customization](#) section for details. If you do this your users will have links to both workflows from the login screen:



A screenshot of a 'Member sign in' form. It features a title 'Member sign in' at the top. Below it are two input fields: 'Username:' and 'Password:'. Under the password field is a checkbox labeled 'Remember me' followed by a link 'Forgot password?'. At the bottom are two buttons: 'Register as new User' and 'Log in'.

5.1. Registration

Navigate to `/register/`:

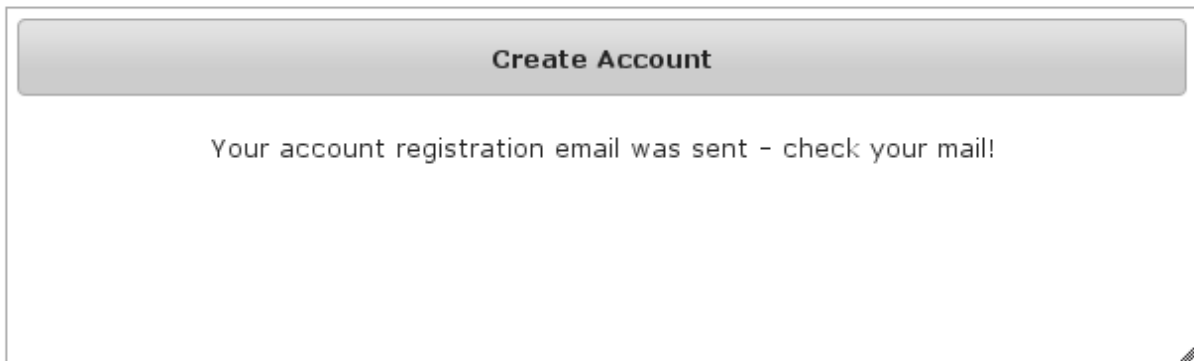


A screenshot of a 'Create Account' form. It has a title bar 'Create Account'. Below it are four input fields: 'Username', 'E-mail', 'Password', and 'Password (again)'. At the bottom is a button labeled 'Create your account'.

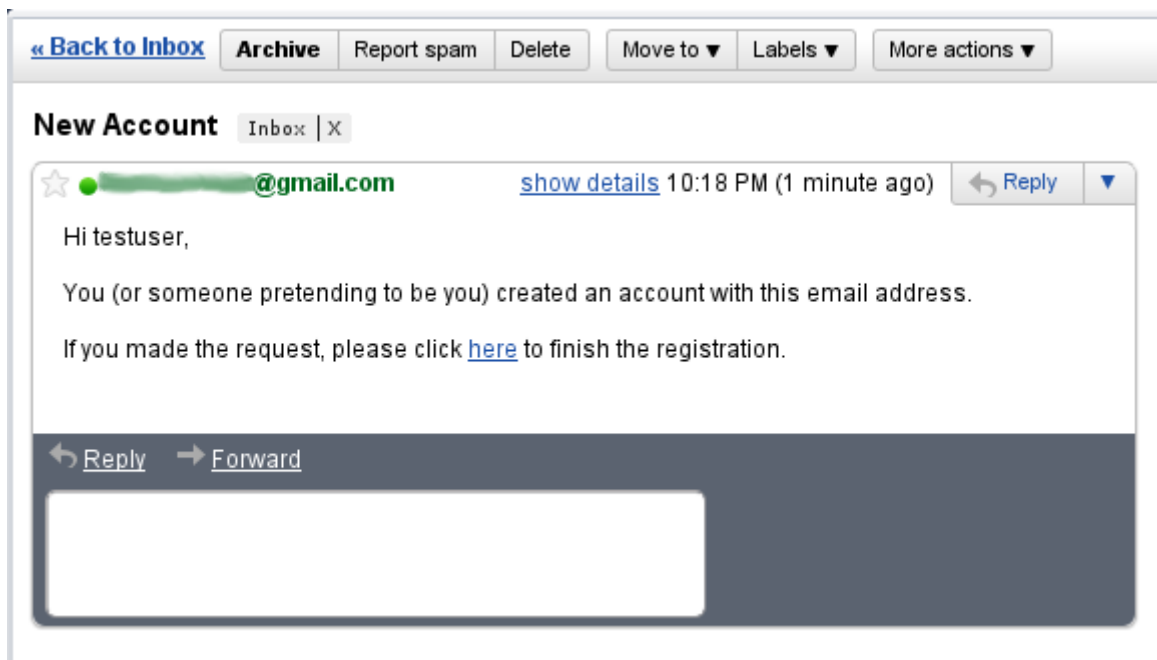
By default e-mail validation for user creation is turned on. If you wish to turn it off see the 'E-Mail Validation Parameter' section below. If this is turned off then no e-mail is sent to the user and they

are automatically redirected to the link that would have been sent in the e-mail.

After filling out valid values an email will be sent and you'll see a success screen:




Click on the link in the email:



and you'll finalize the process, which involves enabling the locked user and pre-authenticating, then redirecting to the configured destination:



 Your registration is complete

Logged in as testuser ([Logout](#))

5.2. Configuration

The post-registration destination url is configurable in `grails-app/conf/application.groovy` using the `postRegisterUrl` attribute:

If you don't specify a value then the `grails.plugin.springsecurity.successHandler.defaultTargetUrl` value will be used, which is `'/'` by default.

In addition, each new user will be granted `ROLE_USER` after finalizing the registration. If you want to change the default role, add more, or grant no roles at all (for example if you want an admin to approve new users and explicitly enable new users) then you can customize that with the `defaultRoleNames` attribute (which is a List of Strings):

```
grails.plugin.springsecurity.ui.register.postRegisterUrl = '/welcome'
```

Chapter 6. E-Mail Validation Parameter

You can customize if the user should get e-mail for validation before they can log in for the first time. To turn off e-mail validation set the parameter `requireEmailValidation` to false, if not set it will default to true. This can be set in `grails-app/conf/application.groovy` or `grails-app/conf/application.yml` under

```
grails.plugin.springsecurity.ui.register.requireEmailValidation = false
```

Please Note if you turn off e-mail validation in production it is strongly recommended to have additional user creation validation

6.1. Mail configuration

By default the plugin uses the [Mail](#) plugin to send emails, but only if it is installed. This is configurable by registering your own `MailStrategy` implementation - see the section on [Configuration](#) for more information. The plugin assumes that the Mail plugin and an SMTP server are already configured.

You can customize the subject, body, and from address of the registration email by overriding the default values in `grails-app/conf/application.groovy`, for example:

```
grails.plugin.springsecurity.ui.register.emailBody = '...'
grails.plugin.springsecurity.ui.register.emailFrom = '...'
grails.plugin.springsecurity.ui.register.emailSubject = '...'
```

The `emailBody` property should be a GString and will have the User domain class instance in scope in the `user` variable, and the generated url to click to finalize the signup in the `url` variable.

```
grails.plugin.springsecurity.ui.register.defaultRoleNames = [] // no roles
```

or

```
grails.plugin.springsecurity.ui.register.defaultRoleNames = ['ROLE_CUSTOMER']
```

6.2. Notes

You should consider the registration code as starter code - every signup workflow will be different, and this should help you get going but is unlikely to be sufficient. You may wish to collect more information than just username and email - first and last name for example. Run `grails s2ui-override register` to copy the registration controller and GSPs into your application to be customized.

If there are unexpected validation errors during registration (which can happen when there is a disconnect between the domain classes and the code in `RegisterController` they will be logged at the `warn` or `error` level, so enable logging to ensure that you see the messages, e.g.

```
...
logger 'grails.plugin.springsecurity.ui.SpringSecurityUiService', WARN
...
```



`RegisterController` and its GSPs assume that your User domain class has an `email` field. Be sure to either rework the workflow (using the `s2ui-override` script) if you don't need an email confirmation step or add an email field.

6.3. RegistrationCode search

The plugin uses its `grails.plugin.springsecurity.ui.RegistrationCode` domain class to store a token associated with the new users' username for use when finishing the registration process after the user clicks the link in the generated email (and also as part of the forgot-password workflow). The plugin includes a controller and GSPs to manage these instances.

The default action for the `RegistrationCode` controller is `search`. By default only the standard fields (`username` and `token`) are available but this is customizable with the `s2ui-override` script - see the [Customization](#) section for details.

You can search by any combination of fields, and both fields have an Ajax autocomplete to assist in finding instances. Leave both fields empty to return all instances.

The screenshot shows the 'Spring Security Management Console' interface. At the top, there is a navigation bar with links: 'Users', 'Roles', 'Requestmaps', 'Registration Code', 'ACL', and 'Security Info'. The 'Registration Code' link is highlighted. On the right side of the navigation bar, it says 'Logged in as admin (Logout)'. Below the navigation bar, the main content area is titled 'Registration Code Search'. It contains two input fields: 'Username:' and 'Token:'. Below these fields is a 'Search' button. The entire form is enclosed in a light gray border.

Searching is case-insensitive and the search string can appear anywhere in the field. Results are shown paginated in groups of 10 and you can click on any header to sort by that field:

Registration Code Search

Username: Token:

Token	Username	Date Created
e81b1e53648a47e6aef31a937154c7cb	registration_test_1	2015-12-19
4a7f88afec3746f7aab2f5d0d8df6d8e	registration_test_1	2015-12-19
c7ac5f23be70495f93e4450a78a27cb4	registration_test_1	2015-12-19
a50e061e0e2f424fb7bc2ff3dae597d	registration_test_1	2015-12-19
d6938ad63c414a69a0da30a8c0619a60	registration_test_2	2015-12-19
4a589c642ea143abb2ecaea57fa0a0cc	registration_test_2	2015-12-19
0a154624f36d42e4aa68991a9477bd04	registration_test_2	2015-12-19
3842a6ae102a431c8e48177c16720713	registration_test_3	2015-12-19
84cefa66465a460c82f46120d9098686	registration_test_3	2015-12-19
fd1e40a7b31f4e8282a2a789135ed21d	registration_test_3	2015-12-19

1

Showing 1 through 10 out of 14.

6.4. RegistrationCode edit

After clicking through to a RegistrationCode you get to the edit page (there are no view pages):

Edit RegistrationCode

Username Token

Date Created 2015-12-19

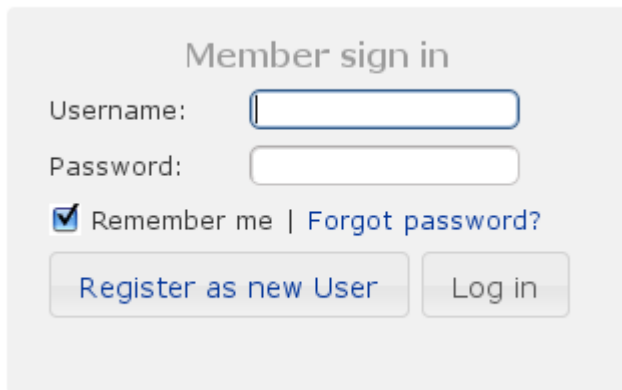
You can update the **username** or **token** attribute or delete the RegistrationCode.

Since instances are created during the "User Registration" and "Forgot Password" workflows, there is no functionality in this plugin to create new instances.

Chapter 7. Forgot Password

Like the Registration workflow, the Forgot Password workflow is expected to be user-facing. So it's not available in the admin menu like the User, Role, and other backend functionality - you'll need to expose them to your users.

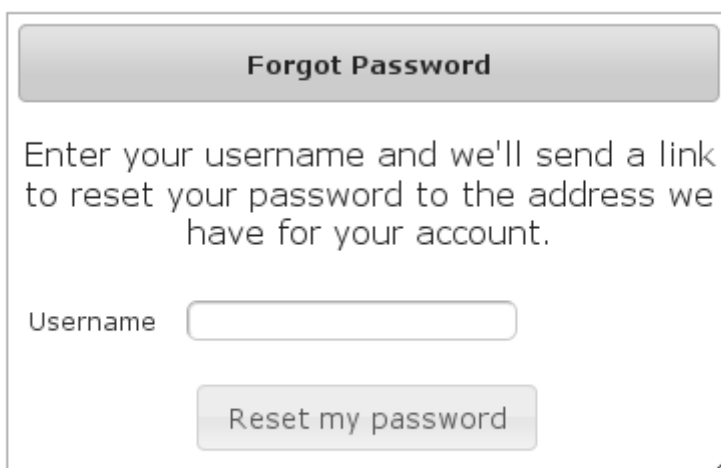
One way to do this is to replace the default `login.gsp` that's provided by the Spring Security Core plugin with this plugin's version. You can do this by running `grails s2ui-override auth` - see the section on [Customization](#) for more details. If you do this your users will have links to both workflows from the login screen:



A screenshot of a 'Member sign in' form. It features a title 'Member sign in' at the top. Below it are two input fields: 'Username:' and 'Password:'. There is a checkbox labeled 'Remember me' which is checked, followed by a link 'Forgot password?'. At the bottom, there are two buttons: 'Register as new User' and 'Log in'.

7.1. Forgot Password

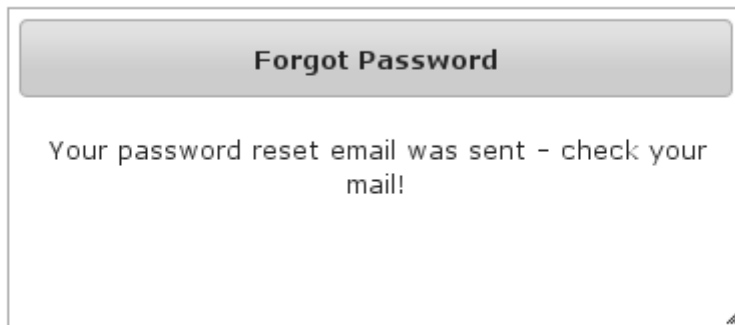
Navigate to `/register/forgotPassword`:



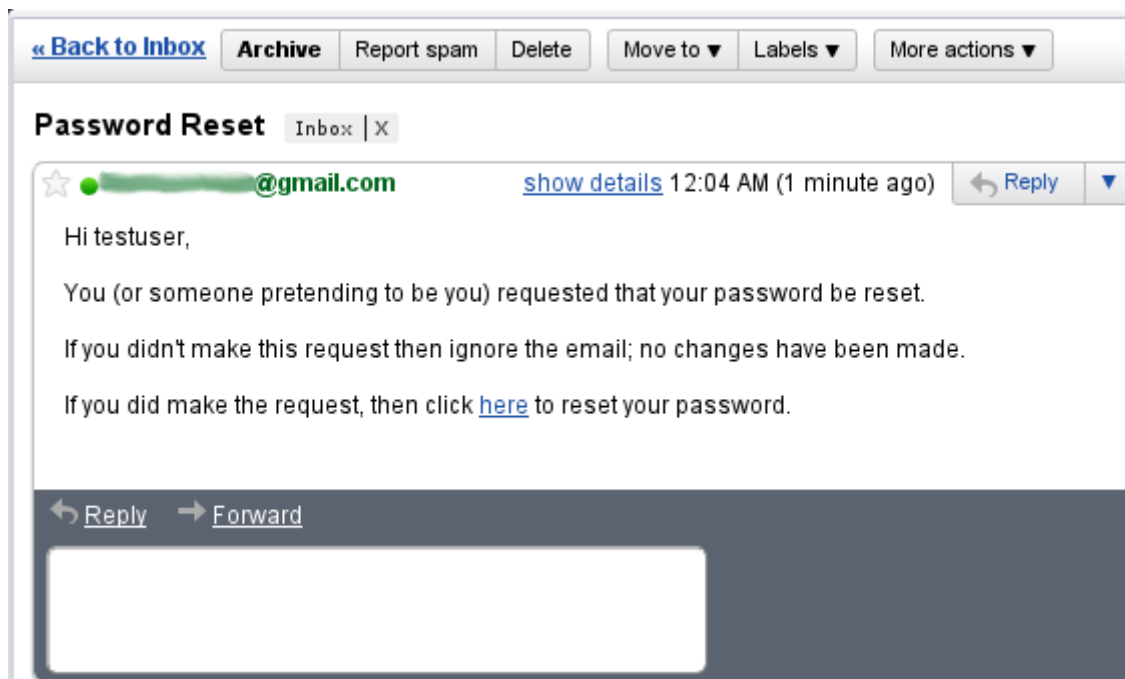
A screenshot of a 'Forgot Password' form. It has a title bar 'Forgot Password'. Below the title bar, it says 'Enter your username and we'll send a link to reset your password to the address we have for your account.' There is a single input field labeled 'Username'. Below the input field is a button labeled 'Reset my password'.

The default is to have e-mail validation for forgetting a password. This is now configurable which will be explained below.

After entering a valid username an email will be sent and you'll see a success screen:



Click on the link in the email:



and you'll open the reset password form:

A rectangular box with a light gray border. At the top is a gray button with the text "Reset Password". Below the button, the text reads: "Enter your new password". There are two input fields: "Password" and "Password (again)". Below the input fields is a gray button with the text "Update my password".

After entering a valid password you'll finalize the process, which involves storing the new

password hashed in the user table and pre-authenticating, then redirecting to the configured destination:



 Your password was successfully changed

Logged in as testuser ([Logout](#))

7.2. Configuration

The post-reset destination url is configurable in `grails-app/conf/application.groovy` using the `postResetUrl` attribute:

```
grails.plugin.springsecurity.ui.forgotPassword.postResetUrl = '/reset'
```

If you don't specify a value then the `defaultTargetUrl` value will be used, which is `'/'` by default.

7.3. Email Validation Configuration

You can customize if you want to have the user get e-mails for validation before they can reset their password. To turn off e-mail validation set the parameter `requireForgotPassEmailValidation` to false. If not set it will default to true. This can be set in `grails-app/conf/application.groovy` or `grails-app/conf/application.yml` under

```
grails.plugin.springsecurity.ui.forgotPassword.requireForgotPassEmailValidation =  
false
```

If e-mail validation is turned off, it is recommended to use the `forgotPasswordExtraValidation` below.

7.4. Challenge Questions Configuration

It is recommended the provided script named `s2ui-create-challenge-questions` be used to generate challenge questions.

All the answers will be encrypted and it will use the same encryption settings as the what is used to store the password for each user. The service to handle this listener is automatically created and configured when you use the plugin.

The option to add challenge questions can be turned on will work independent of e-mail validation. If the e-mail validation is turned on this step will occur before the e-mail is sent out.

To make this work, the domain object must have a link to your User Object. This is done by setting the `validationUserLookupProperty` which defaults to 'user'.

This can be customized in either `grails-app/conf/application.groovy` or `grails-app/conf/application.yml`

```
grails.plugin.springsecurity.ui.forgotPassword.validationUserLookupProperty = 'user'
```

Each list item in the `forgotPasswordExtraValidation` has three options, though each should only have two. If you have both `labelDomain` and `labelMessage` then `labelDomain` will be used as it takes precedences.

Table 1. `forgotPasswordExtraValidation` List items

Header 1	Header 2
labelDomain	This is the property in the domain object that will used to ask the question
labelMessage	If labelDomain is not present then this can be used as a message property to ask a question (ie if your questions are static).
prop	This is used as the property of the domain object to determine if the answer is correct.

This can be customized in either `grails-app/conf/application.groovy` or `grails-app/conf/application.yml`

grails-app/conf/application.yml

```
grails:
  plugin:
    springsecurity:
      ui:
        forgotPassword:
          forgotPasswordExtraValidation:
            -
              labelDomain: myQuestion
              prop: myAnswer
            -
              labelMessage: securityvalidations.labelMessage.label1
              prop: myAnswer2
```

If the above is configured you will see a menu item to List and Create Challenge questions for each user.

To use the challenge questions functionality please set the domain class which will hold the answers with the `forgotPasswordExtraValidationDomainClassName` property. This is Domain object that contains any reference that is needed by prop (question) and labelDomain (answer).


```
grails:
  plugin:
    springsecurity:
      ui:
        forgotPassword:
          forgotPasswordExtraValidationDomainClassName: com.mycompany.Profile
```

7.5. Mail configuration

By default the plugin uses the [Mail](#) plugin to send emails, but only if it is installed. This is configurable by registering your own [MailStrategy](#) implementation - see the section on [Configuration](#) for more information. The plugin assumes that the Mail plugin and an SMTP server are already configured.

You can customize the subject, body, and from address of the reset email by overriding the default values in `grails-app/conf/application.groovy`, for example:

```
grails.plugin.springsecurity.ui.forgotPassword.emailBody = '...'
grails.plugin.springsecurity.ui.forgotPassword.emailFrom = '...'
grails.plugin.springsecurity.ui.forgotPassword.emailSubject = '...'
```

The `emailBody` property should be a GString and will have the User domain class instance in scope in the `user` variable, and the generated url to click to reset the password in the `url` variable.

7.6. Notes

Like the registration code, consider this workflow as starter code. Run `grails s2ui-override register` to copy the registration controller and GSPs into your application to be customized.



`RegisterController` and its GSPs assume that your User domain class has an `email` field.

Chapter 8. ACL Management


ACL management should be done using the API exposed by `AclService` and `AclUtilService`. Both services have a much more intuitive and convenient high-level approach to managing ACLs, ACEs, etc. The functionality in this plugin is to provide a CRUD interface for fine-grained ACL management.

The ACL menu is only available if the [ACL plugin](#) is installed.

8.1. AclClass Management

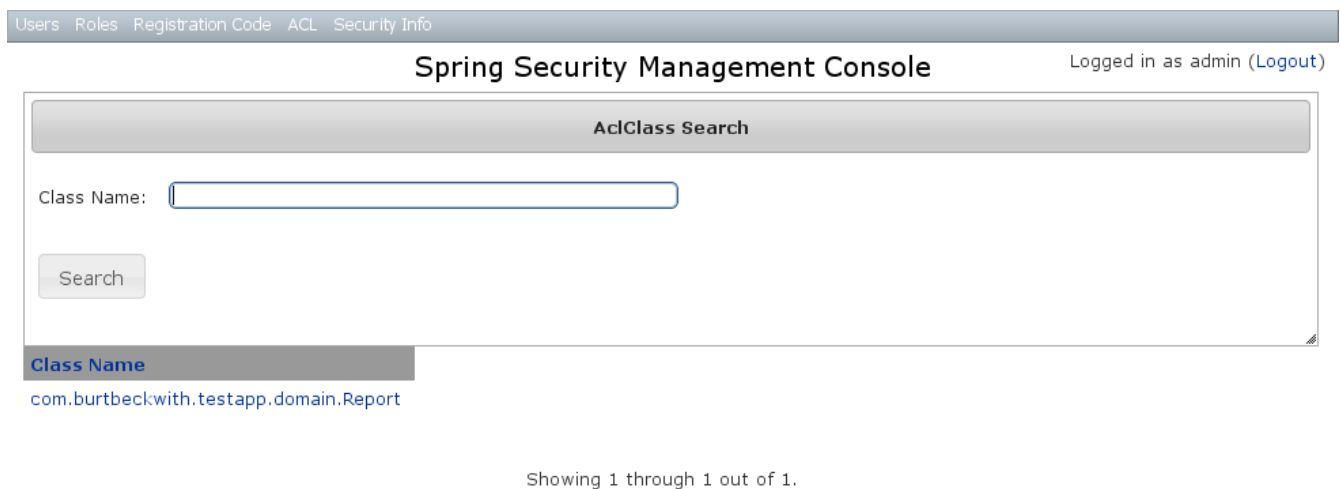
The default action for the AclClass controller is search. By default only the standard fields are available but this is customizable with the `s2ui-override` script - see the [Customization](#) section for details.

The `className` field has an Ajax autocomplete to assist in finding instances. Leave the field empty to return all instances.



The screenshot shows the top navigation bar with links: Users, Roles, Registration Code, Security Info. The main header reads "Spring Security Management Console" and "Logged in as admin (Logout)". Below this is a form titled "AclClass Search". It contains a "Class Name:" label followed by a text input field. Below the input field is a "Search" button.

Searching is case-insensitive and the search string can appear anywhere in the field. Results are shown paginated in groups of 10 and you can click on the `className` column header to sort the results by that field:



This screenshot shows the same "AclClass Search" form, but with results displayed below it. The "Class Name" column header is highlighted in blue. Below it, a single result is shown: `com.burtbeckwith.testapp.domain.Report`. At the bottom, it says "Showing 1 through 1 out of 1."

8.1.1. AclClass Edit

After clicking through an AclClass you get to the edit page (there are no view pages):

Edit AclClass

Class Name

[View Associated OIDs](#)

[View Associated ACL Entries](#)

You can update the name, and delete the instance if there aren't any associated `AclObjectIdentity` or `AclEntry` instances - by default there is no support for cascading.

You can also see the associated `AclObjectIdentity` instances (OIDs) or `AclEntry` instances.

8.1.2. AclClass Create

You can create new instances by going to `/aclClass/create` or by clicking the `Create` action in the `Class` menu under `ACL`.

Create AclClass

Class Name

8.2. AclSid Management

The default action for the `AclSid` controller is search. By default only the standard fields are available but this is customizable with the `s2ui-override` script - see the [Customization](#) section for details.

The `sid` field has an Ajax autocomplete to assist in finding instances. Leave the field empty and `principal` set to `Either` to return all instances.

AclSid Search

SID:

☐ True
 ☐ False
 ☒ Either

Principal: ☐ ☐ ☒

Results are shown paginated in groups of 10. The column headers are clickable and will sort the results by that field:

AclSid Search

SID:

☐ True
 ☐ False
 ☒ Either

Principal: ☐ ☐ ☒

SID	Principal
user2	True
admin	True
user1	True

Showing 1 through 3 out of 3.

8.2.1. AclSid Edit

After clicking through to a sid you get to the edit page (there are no view pages):

Edit AclSid

SID

Principal ☒

[View Associated OIDs](#)

[View Associated ACL Entries](#)

You can update the name and whether it's a Principal sid or a Role sid, and delete the instance if there aren't any associated **AclObjectIdentity** or **AclEntry** instances - by default there is no support

for cascading.

You can also see the associated [AclObjectIdentity](#) instances (OIDs) or [AclEntry](#) instances.

8.2.2. AclSid Create

You can create new instances by going to [/aclSid/create](#) or by clicking the [Create](#) action in the [SID](#) menu under [ACL](#).

The screenshot shows the 'Create AclSid' form in the Spring Security Management Console. The page has a navigation bar with links: Users, Roles, Registration Code, ACL, and Security Info. The user is logged in as 'admin' with a 'Logout' link. The form itself has a title bar 'Create AclSid'. It contains a text input field for 'SID', a checkbox for 'Principal', and a 'Create' button.

8.3. AclObjectIdentity Management

The default action for the [AclObjectIdentity](#) controller is search. By default only the standard fields are available but this is customizable with the [s2ui-override](#) script - see the [Customization](#) section for details.

Leave all fields at their default values to return all instances.

The screenshot shows the 'AclObjectIdentity Search' form in the Spring Security Management Console. The page has the same navigation bar and user information as the previous screenshot. The form has a title bar 'AclObjectIdentity Search'. It contains several search criteria: 'AclClass' with a dropdown menu set to 'All', 'Object ID' with a text input field, 'Owner' with a dropdown menu set to 'All', 'Parent' with a text input field, and 'Entries Inheriting' with three radio buttons: 'True' (unchecked), 'False' (unchecked), and 'Either' (checked). There is a 'Search' button at the bottom left.

Results are shown paginated in groups of 10 and you can click on any header to sort by that field:

AclObjectIdentity Search

AclClass: All
Object ID:
Owner: All
Parent:

True
False
Either

Entries Inheriting: ☐ True ☐ False ☒ Either

Search

ID	AclClass	Object ID	Entries Inheriting	Owner	Parent
3	com.burtbeckwith.testapp.domain.Report	3	True	admin	
6	com.burtbeckwith.testapp.domain.Report	6	True	admin	
1	com.burtbeckwith.testapp.domain.Report	1	True	user1	
4	com.burtbeckwith.testapp.domain.Report	4	True	admin	
10	com.burtbeckwith.testapp.domain.Report	10	True	admin	
8	com.burtbeckwith.testapp.domain.Report	8	True	admin	
7	com.burtbeckwith.testapp.domain.Report	7	True	admin	
5	com.burtbeckwith.testapp.domain.Report	5	True	admin	
9	com.burtbeckwith.testapp.domain.Report	9	True	admin	
2	com.burtbeckwith.testapp.domain.Report	2	True	user1	

1 2 3 4 5 6 7 8 9 10 Next

Showing 1 through 10 out of 100.

8.3.1. AclObjectIdentity Edit

After clicking through to an AclObjectIdentity you get to the edit page (there are no view pages):

Edit AclObjectIdentity

AclClass com.burtbeckwith.testapp.domain.Report
Object ID 3
Owner admin
Parent
Entries Inheriting ☒

[View Associated ACL Entries](#)

Update
Delete

You can update any of the attributes, and can delete the instance if there aren't any associated **AclEntry** instances - by default there is no support for cascading.

You can also see the associated **AclEntry** instances.

8.3.2. AclObjectIdentity Create

You can create new instances by going to </aclObjectIdentity/create> or by clicking the **Create** action in the **OID** menu under **ACL**.

[Users](#) [Roles](#) [Registration Code](#) [ACL](#) [Security Info](#)

Spring Security Management Console Logged in as admin ([Logout](#))

Create AclObjectIdentity

AclClass

Object ID

Owner

Parent

Entries Inheriting

☐

Create

8.4. AclEntry Management

The default action for the AclEntry controller is search. By default only the standard fields are available but this is customizable with the [s2ui-override](#) script - see the [Customization](#) section for details.

Leave all fields at their default values to return all instances.

[Users](#) [Roles](#) [Registration Code](#) [ACL](#) [Security Info](#)

Spring Security Management Console Logged in as admin ([Logout](#))

AclEntry Search

AclObjectIdentity:

Ace Order:

SID:

All

Mask:

Granting:

☐

☐

☒

Audit Success:

☐

☐

☒

Audit Failure:

☐

☐

☒

Search

Results are shown paginated in groups of 10 and you can click on any header to sort by that field:

AclEntry Search

AclObjectIdentity:

Ace Order:

SID:

All

Mask:

True

False

Either

Granting:

☐

☐

☒

Audit Success:

☐

☐

☒

Audit Failure:

☐

☐

☒

ID	AclObjectIdentity	Ace Order	SID	Mask	Granting	Audit Success	Audit Failure
99	5	2	user2	BasePermission[.....W.=2]	True	False	False
94	4	0	user1	BasePermission[.....R=1]	True	False	False
95	4	1	user2	BasePermission[.....R=1]	True	False	False
100	5	3	admin	BasePermission[.....A...=16]	True	False	False
96	4	2	admin	BasePermission[.....A...=16]	True	False	False
93	3	2	admin	BasePermission[.....A...=16]	True	False	False
91	3	0	user1	BasePermission[.....R=1]	True	False	False
92	3	1	user2	BasePermission[.....R=1]	True	False	False
97	5	0	user1	BasePermission[.....R=1]	True	False	False
98	5	1	user2	BasePermission[.....R=1]	True	False	False

1

2

3

4

5

6

7

8

9

10

..

18

Showing 1 through 10 out of 175.

8.4.1. AclEntry Edit

After clicking through to an AclEntry you get to the edit page (there are no view pages):

Edit AclEntry

AclObjectIdentity

Ace Order

SID

user2

Mask

Granting

☒

Audit Success

☐

Audit Failure

☐

You can update any of the attributes or delete the AclEntry.

8.4.2. AclEntry Create

You can create new instances by going to </aclEntry/create> or by clicking the **Create** action in the

Entry menu under ACL.

Create AclEntry

AclObjectIdentity

Ace Order

0

SID

Mask

0

Granting

☒

Audit Success

☐

Audit Failure

☐

Create

Chapter 9. Persistent Cookie Management

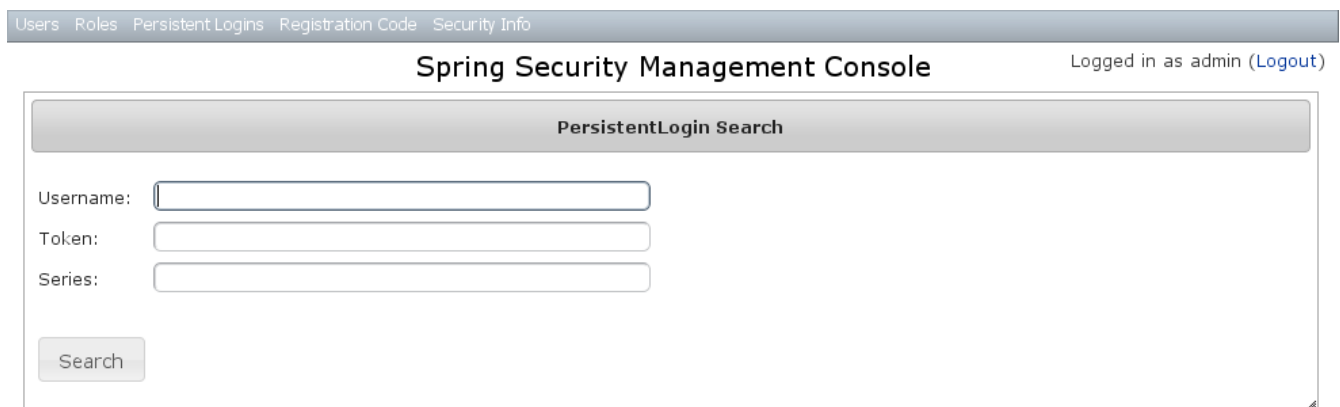
Persistent cookies aren't enabled by default - you must enable them by running the `s2-create-persistent-token` script. See [the Spring Security Core plugin documentation](#) for details about this feature.

The Persistent Logins menu is only shown if this feature is enabled.

9.1. Persistent logins search

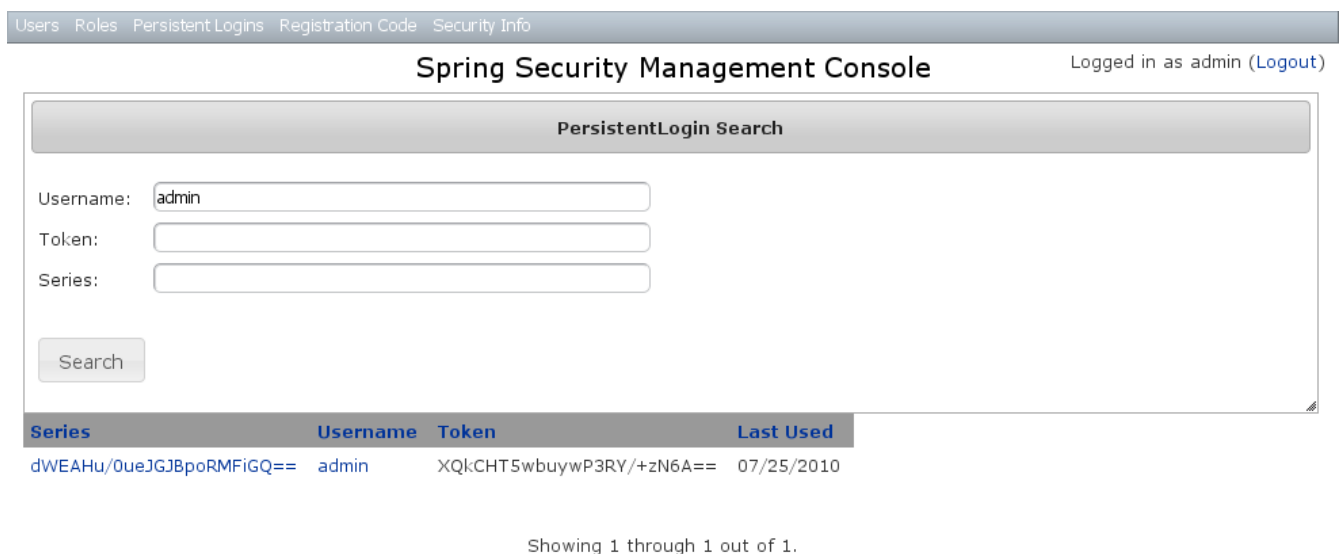
The default action for the PersistentLogin controller is search. By default only the standard fields (`username`, `token`, and `series`) are available but this is customizable with the `s2ui-override` script - see the [Customization](#) section for details.

You can search by any combination of fields, and all fields have an Ajax autocomplete to assist in finding instances. Leave all fields empty to return all instances.



The screenshot shows the Spring Security Management Console interface. At the top, there is a navigation bar with links: Users, Roles, Persistent Logins, Registration Code, and Security Info. The title "Spring Security Management Console" is centered, and on the right, it says "Logged in as admin (Logout)". Below this is a section titled "PersistentLogin Search". It contains three input fields: "Username:", "Token:", and "Series:". A "Search" button is located below the "Series" field.

Searching is case-insensitive and the search string can appear anywhere in the field. Results are shown paginated in groups of 10 and you can click on any header to sort by that field:



The screenshot shows the same Spring Security Management Console interface, but now with search results. The "Username" field is filled with "admin". Below the search fields, there is a table with the following data:

Series	Username	Token	Last Used
dWEAHu/0ueJGJBpoRMFiGQ==	admin	XQkCHT5wbuywP3RY/+zN6A==	07/25/2010

Below the table, it says "Showing 1 through 1 out of 1."

9.2. Persistent logins edit

After clicking through to an instance you get to the edit page (there are no view pages):

[Users](#) [Roles](#) [Persistent Logins](#) [Registration Code](#) [Security Info](#)

Spring Security Management Console Logged in as admin ([Logout](#))

Edit PersistentLogin

Series

dWEAHu/0ueJGJBpoRMFiGQ==

Username

admin

Token

Last Used

Update

Delete

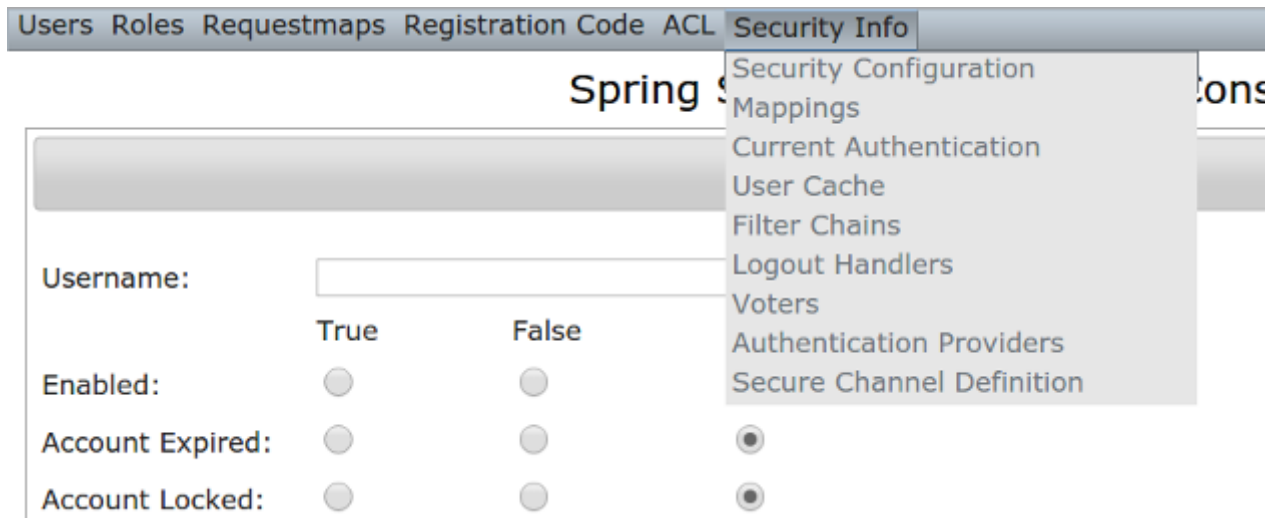
You can update the **token** or **lastUsed** attribute or delete the instance.

9.3. Persistent logins creation

Since instances are created during authentication by the spring-security-core plugin, there is no functionality in this plugin to create new instances.

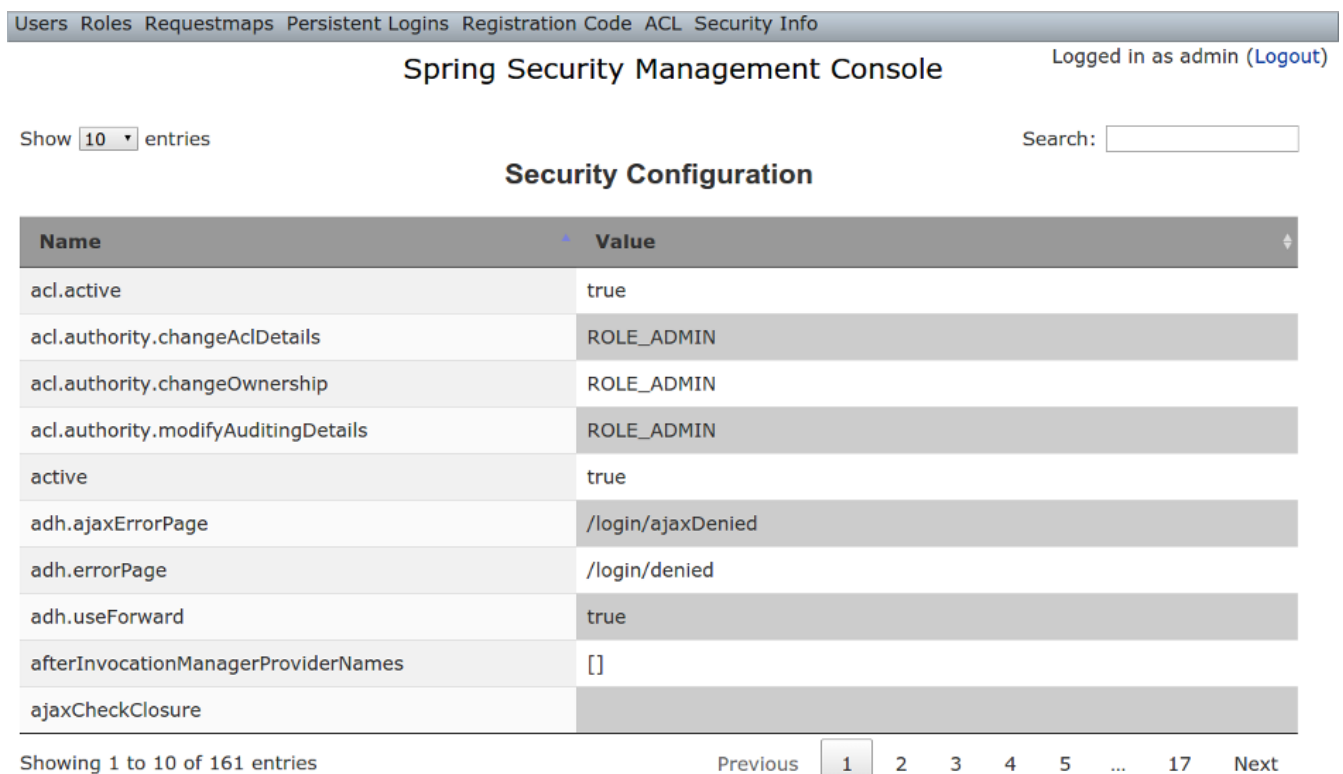
Chapter 10. Security Configuration UI

The Security Info menu has links for several pages that contain read-only views of much of the Spring Security configuration:



10.1. Security Configuration

The Security Configuration menu item displays all security-related attributes in `application.groovy`. The names omit the `grails.plugin.springsecurity` prefix:



10.2. Mappings

The Mappings menu item displays the current request mapping mode (Annotation, Requestmap, or Static) and all current mappings:

SecurityConfigType: Annotation

Mappings

Pattern	ConfigAttributes	HTTP Method
/	permitAll	all
/index	permitAll	all
/index.gsp	permitAll	all
/assets/**	permitAll	all
/**/js/**	permitAll	all
/**/css/**	permitAll	all
/**/images/**	permitAll	all
/**/favicon.ico	permitAll	all
/register	permitAll	all
/register/**	permitAll	all
/registrationcode	permitAll	all
/registrationcode/**	permitAll	all
/securityinfo	permitAll	all
/securityinfo/**	permitAll	all
/login	permitAll	all
/login.*	permitAll	all
/login/**	permitAll	all
/logout	permitAll	all
/logout.*	permitAll	all
/logout/**	permitAll	all

10.3. Current Authentication

The Current Authentication menu item displays your **Authentication** information, mostly for reference to see what a typical one contains:

Current Authentication

Name	Value
Authorities	[ROLE_RUN_AS, ROLE_USER, ROLE_ADMIN, ROLE_SWITCH_USER]
Details	org.springframework.security.web.authentication.WebAuthenticationDetails@ffed504: RemoteIpAddress: 127.0.0.1; SessionId: 659C5EEAED7F26774E7214E7F0D35D3D
Principal	grails.plugin.springsecurity.userdetails.GrailsUser@586034f: Username: admin; Password: [PROTECTED]; Enabled: true; AccountNonExpired: true; credentialsNonExpired: true; AccountNonLocked: true; Granted Authorities: ROLE_ADMIN,ROLE_RUN_AS,ROLE_SWITCH_USER,ROLE_USER
Name	admin

10.4. User Cache

The User Cache menu item displays information about cached users if the feature is enabled (it is disabled by default).

UserCache class: net.sf.ehcache.Cache

User Cache

Attribute	Value
Size	1
Status	STATUS_ALIVE
Name	userCache
GUID	127.0.1.1-458656cf-0031-4216-8f6d-ee25a6438d98

Statistics

Attribute	Value
Cache Hits	1
In-memory Hits	2
On-disk Hits	0
Cache Misses	3
Object Count	1
Memory Store Object Count	1
Disk Store Object Count	0
Eviction Count	0

1 Cached User(s)

Username	User
admin	grails.plugin.springsecurity.userdetails.GrailsUser@586034f: Username: admin; Password: [PROTECTED]; Enabled: true; AccountNonExpired: true; credentialsNonExpired: true; AccountNonLocked: true; Granted Authorities: ROLE_ADMIN,ROLE_RUN_AS,ROLE_SWITCH_USER,ROLE_USER

10.5. Filter Chains

The Filter Chains menu item displays your configured Filter chains. It is possible to have multiple URL patterns each with its own filter chain, for example when using HTTP Basic Auth for a web service. By default since the 3.0.0 release the spring-security-core [s2-quickstart](#) script configures empty filter chains for static assets to avoid unnecessary security checks (although of course if you need to secure some or all of your static assets you should reconfigure these).

Filter Chains

URL Pattern	Filters
Ant [pattern='/assets/**']	none
Ant [pattern='/**/*.js/**']	none
Ant [pattern='/**/*.css/**']	none
Ant [pattern='/**/*.images/**']	none
Ant [pattern='/**/*.favicon.ico']	none
Ant [pattern='/**']	grails.plugin.springsecurity.web.SecurityRequestHolderFilter org.springframework.security.web.access.channel.ChannelProcessingFilter org.springframework.security.web.context.SecurityContextPersistenceFilter grails.plugin.springsecurity.web.authentication.logout.MutableLogoutFilter grails.plugin.springsecurity.web.authentication.GrailsUsernamePasswordAuthenticationFilter org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter grails.plugin.springsecurity.web.filter.GrailsRememberMeAuthenticationFilter grails.plugin.springsecurity.web.filter.GrailsAnonymousAuthenticationFilter org.springframework.security.web.access.ExceptionTranslationFilter org.springframework.security.web.access.intercept.FilterSecurityInterceptor org.springframework.security.web.authentication.switchuser.SwitchUserFilter

10.6. Logout Handlers

The Logout Handlers menu item displays your registered `LogoutHandlers`. Typically there will be just the ones shown here, but you can register your own custom implementations, or a plugin might contribute more:

Users	Roles	Requestmaps	Persistent Logins	Registration Code	ACL	Security Info	
Spring Security Management Console							Logged in as admin (Logout)
Logout Handlers							
Class Name							
org.springframework.security.web.authentication.rememberme.PersistentTokenBasedRememberMeServices							
org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler							

10.7. Voters

The Voters menu item displays your registered `AccessDecisionVoters`. Typically there will be just the ones shown here, but you can register your own custom implementations, or a plugin might contribute more:

Users	Roles	Requestmaps	Persistent Logins	Registration Code	ACL	Security Info	
Spring Security Management Console							Logged in as admin (Logout)
Voters							
Class Name							
org.springframework.security.access.vote.AuthenticatedVoter							
org.springframework.security.access.vote.RoleHierarchyVoter							
grails.plugin.springsecurity.web.access.expression.WebExpressionVoter							
grails.plugin.springsecurity.access.vote.ClosureVoter							

10.8. Authentication Providers

The Authentication Providers menu item displays your registered `AuthenticationProviders`. Typically there will be just the ones shown here, but you can register your own custom implementations, or a plugin (e.g. LDAP) might contribute more:

Users	Roles	Requestmaps	Persistent Logins	Registration Code	ACL	Security Info	
Spring Security Management Console							Logged in as admin (Logout)
Authentication Providers							
Class Name							
org.springframework.security.authentication.dao.DaoAuthenticationProvider							
grails.plugin.springsecurity.authentication.GrailsAnonymousAuthenticationProvider							
org.springframework.security.authentication.RememberMeAuthenticationProvider							

10.9. Secure Channel Definition

The Secure Channel Definition menu item displays your registered channel security mappings.

Users Roles Persistent Logins Registration Code ACL Security Info	
Spring Security Management Console	
Logged in as admin (Logout)	
Secure Channel Definition	
Pattern	ConfigAttributes
Ant [pattern='/secure/stuff/**']	[REQUIRES_SECURE_CHANNEL]
Ant [pattern='/insecure/stuff/**']	[REQUIRES_INSECURE_CHANNEL]
Ant [pattern='/**']	[ANY_CHANNEL]

Chapter 11. Customization

Most aspects of the plugin are configurable.

11.1. s2ui-override script

The plugin's controllers and GSPs are easily overridden using the `s2ui-override` script. The general syntax for running the script is

```
grails s2ui-override <type> [<controller-package>]
```

The script will copy an empty controller that extends the corresponding plugin controller into your application so you can override individual actions and methods as needed. It also copies the controller's GSPs. The exceptions are 'auth' and 'layout' which only copy GSPs.

For `auth` and `layout`, you do not supply a package name.

For example:

```
grails s2ui-override auth
```

For the other types `aclclass`, `aclentry`, `aclobjectidentity`, `aclsid`, `persistentlogin`, `register`, `registrationcode`, `requestmap`, `role`, `securityinfo`, `user`, you supply a package name as command parameter. That it is to say:

```
grails s2ui-override persistentlogin com.company.myapp
```

The files copied for each type are summarized here:

- `aclclass`
 - `controller/AclClassController.groovy`
 - `views/aclClass/create.gsp`
 - `views/aclClass/edit.gsp`
 - `views/aclClass/search.gsp`
- `aclentry`
 - `controller/AclEntryController.groovy`
 - `views/aclEntry/create.gsp`
 - `views/aclEntry/edit.gsp`
 - `views/aclEntry/search.gsp`
- `aclobjectidentity`
 - `controller/AclObjectIdentityController.groovy`
 - `views/aclObjectIdentity/create.gsp`

- `views/aclObjectIdentity/edit.gsp`
- `views/aclObjectIdentity/search.gsp`
- `aclsid`
 - `controller/AclSidController.groovy`
 - `views/aclSid/create.gsp`
 - `views/aclSid/edit.gsp`
 - `views/aclSid/search.gsp`
- `auth`
 - `views/login/auth.gsp`
- `layout`
 - `views/layouts/springSecurityUI.gsp`
 - `views/includes/_ajaxLogin.gsp`
- `persistentlogin`
 - `controller/PersistentLoginController.groovy`
 - `views/persistentLogin/edit.gsp`
 - `views/persistentLogin/search.gsp`
- `register`
 - `controller/RegisterController.groovy`
 - `views/layouts/email.gsp`
 - `views/register/forgotPassword.gsp`
 - `views/register/_forgotPasswordMail.gsp`
 - `views/register/register.gsp`
 - `views/register/resetPassword.gsp`
 - `views/register/_verifyRegistrationMail.gsp`
- `registrationcode`
 - `controller/RegistrationCodeController.groovy`
 - `views/registrationCode/edit.gsp`
 - `views/registrationCode/search.gsp`
- `requestmap`
 - `controller/RequestmapController.groovy`
 - `views/requestmap/create.gsp`
 - `views/requestmap/edit.gsp`
 - `views/requestmap/search.gsp`
- `role`
 - `controller/RoleController.groovy`
 - `views/role/create.gsp`
 - `views/role/edit.gsp`

- `views/role/search.gsp`
- `securityinfo`
 - `controller/SecurityInfoController.groovy`
 - `views/securityInfo/config.gsp`
 - `views/securityInfo/currentAuth.gsp`
 - `views/securityInfo/filterChains.gsp`
 - `views/securityInfo/logoutHandlers.gsp`
 - `views/securityInfo/mappings.gsp`
 - `views/securityInfo/providers.gsp`
 - `views/securityInfo/secureChannel.gsp`
 - `views/securityInfo/usercache.gsp`
 - `views/securityInfo/voters.gsp`
- `user`
 - `controller/UserController.groovy`
 - `views/user/create.gsp`
 - `views/user/edit.gsp`
 - `views/user/search.gsp`

11.2. I18N

All of the plugin's displayed strings are localized and stored in the plugin's `grails-app/i18n/messages.spring-security-ui.properties` file. You can override any of these values by putting an override in your application's `grails-app/i18n/messages.properties` file.

11.3. application.groovy attributes

There are a few configuration options specified in `DefaultUiSecurityConfig.groovy` that can be overridden in your application's `grails-app/conf/application.groovy`

11.3.1. Registration attributes

These settings are used in the registration workflow; see the [User Registration](#) section for more details:

- `grails.plugin.springsecurity.ui.register.defaultRoleNames`
- `grails.plugin.springsecurity.ui.register.emailBody`
- `grails.plugin.springsecurity.ui.register.emailFrom`
- `grails.plugin.springsecurity.ui.register.emailSubject`
- `grails.plugin.springsecurity.ui.register.postRegisterUrl`

11.3.2. Forgot Password attributes

These settings are used in the forgot-password workflow; see the [Forgot Password](#) section for more details:

- `grails.plugin.springsecurity.ui.forgotPassword.emailBody`
- `grails.plugin.springsecurity.ui.forgotPassword.emailFrom`
- `grails.plugin.springsecurity.ui.forgotPassword.emailSubject`
- `grails.plugin.springsecurity.ui.forgotPassword.postResetUrl`

11.3.3. GSP layout attributes

The `layout` attribute in the GSPs is configurable. If this is the only change you want to make in some or all of the GSPs then you can avoid copying the GSPs into your application just to make this change.

The default value for the registration workflow GSPs (`forgotPassword.gsp`, `register.gsp`, and `resetPassword.gsp`) is “register” and the default for the rest is “springSecurityUI”. These values can be overridden with the `grails.plugin.springsecurity.ui.gsp.layoutRegister` and `grails.plugin.springsecurity.ui.gsp.layoutUi` settings.

11.3.4. Miscellaneous attributes

The role name required to be able to run as another user defaults to `ROLE_SWITCH_USER` but you can override this name with the `grails.plugin.springsecurity.ui.switchUserRoleName` setting.

11.4. CSS and JavaScript

The plugin uses the [Asset Pipeline](#) plugin to manage its resources. This makes it very easy to provide your own version of some or all of the static resources since asset-pipeline will always use a file in the application’s `assets` directory instead of a plugin’s if it exists.

Instead of depending on either the jQuery or jQuery UI plugins, this plugin includes its own copy of `jquery.js`, `jquery-ui.js`, and `jquery-ui.css`. Note that the versions are not hard-coded, but instead they take advantage of the feature in asset-pipeline where you can embed Groovy code in a file to specify the name and path.

The layouts use `grails-app/assets/javascripts/jquery.js`, which contains this:

```
//require jquery/jquery-  
${grails.plugin.springsecurity.ui.Constants.JQUERY_VERSION}.js
```

This resolves to `grails-app/assets/javascripts/jquery/jquery-2.1.4.js`, and to use your own version, either use the same approach in a file called `jquery.js` or rename your file to `jquery.js`.

Likewise for jQuery UI, the JavaScript file is `grails-app/assets/javascripts/jquery-ui.js`, which contains this

```
//=require jquery-ui/jquery-ui-  
${grails.plugin.springsecurity.ui.Constants.JQUERY_UI_VERSION}.js
```

and the CSS file `grails-app/assets/stylesheets/jquery-ui.css`, which contains

```
/*  
*= require smoothness/jquery-ui-  
${grails.plugin.springsecurity.ui.Constants.JQUERY_UI_VERSION}.css  
*/
```

The JavaScript file resolves to `grails-app/assets/javascripts/jquery-ui/jquery-ui-1.10.3.custom.js`, and to use your own version, either use the same approach in a file called `jquery-ui.js` or rename your file to `jquery-ui.js`.

The CSS file resolves to `grails-app/assets/stylesheets/smoothness/jquery-ui-1.10.3.custom.css`, and to use your own version, either use the same approach in a file called `jquery-ui.js` or rename your file to `jquery-ui.js`.

Use your own `jquery-ui.js` and/or `jquery-ui.css` to override the plugin's.

The `springSecurityUI.gsp` layout includes `grails-app/assets/stylesheets/spring-security-ui.css`, which has no style declarations and only includes other CSS files:

```
/*  
*= require reset.css  
*= require jquery-ui.css  
*= require jquery.jdMenu.css  
*= require jquery.jdMenu.slate.css  
*= require jquery.jgrowl.css  
*= require spring-security-ui-common.css  
*/
```

and `grails-app/assets/javascripts/spring-security-ui.js` which has no JavaScript code and only includes other JavaScript files:

```
//= require jquery.js  
//= require jquery-ui.js  
//= require jquery/jquery.jgrowl.js  
//= require jquery/jquery.positionBy.js  
//= require jquery/jquery.bgiframe.js  
//= require jquery/jquery.jdMenu.js  
//= require jquery/jquery.form.js  
//= require spring-security-ui-ajaxLogin.js
```

The `register.gsp` layout includes `grails-app/assets/stylesheets/spring-security-ui-register.css`, which has no style declarations and only includes other CSS files:

```
/*
*= require reset.css
*= require jquery-ui.css
*= require jquery.jgrowl.css
*= require spring-security-ui-common.css
*/
```

and `grails-app/assets/javascripts/spring-security-ui-register.js` which has no JavaScript code and only includes other JavaScript files:

```
//= require jquery.js
//= require jquery-ui.js
//= require jquery/jquery.jgrowl.js
```

The remaining JavaScript files are

- `grails-app/assets/javascripts/spring-security-ui-ajaxLogin.js`
- `grails-app/assets/javascripts/jquery/jquery.bgiframe.js`
- `grails-app/assets/javascripts/jquery/jquery.dataTables.js`
- `grails-app/assets/javascripts/jquery/jquery.form.js`
- `grails-app/assets/javascripts/jquery/jquery.jdMenu.js`
- `grails-app/assets/javascripts/jquery/jquery.jgrowl.js`
- `grails-app/assets/javascripts/jquery/jquery.positionBy.js`

and the remaining CSS files are

- `grails-app/assets/stylesheets/jquery.dataTables.css`
- `grails-app/assets/stylesheets/jquery.jdMenu.css`
- `grails-app/assets/stylesheets/jquery.jdMenu.slate.css`
- `grails-app/assets/stylesheets/jquery.jgrowl.css`
- `grails-app/assets/stylesheets/reset.css`
- `grails-app/assets/stylesheets/spring-security-ui-auth.css`
- `grails-app/assets/stylesheets/spring-security-ui-common.css`

11.5. Password Hashing

In recent versions of the Spring Security Core plugin, the “User” domain class is generated by the `s2-quickstart` script with code to automatically hash the password. This makes the code simpler (for example in controllers where you create users or update user passwords) but older generated classes don’t have this generated code. This presents a problem for plugins like this one since it’s not possible to reliably determine if the domain class hashes the password or if you use the older approach of explicitly calling `springSecurityService.encodePassword()`.

The unfortunate consequence of mixing a newer domain class that does password hashing with controllers that call `springSecurityService.encodePassword()` is the the passwords get double-hashed, and users aren't able to login. So to get around this there's a configuration option you can set to tell this plugin's controllers whether to hash or not: `grails.plugin.springsecurity.ui.encodePassword`.

This option defaults to `false`, so if you have an older domain class that doesn't handle hashing just enable this plugin's hashing:

```
grails.plugin.springsecurity.ui.encodePassword = true
```

h4. Strategy classes

The plugin's `SpringSecurityUiService` implements several "strategy" interfaces to make it possible to override its functionality in a more fine-grained way.

These are defined by interfaces in the `grails.plugin.springsecurity.ui.strategy` package:

- `AcLStrategy`
- `ErrorsStrategy`
- `MailStrategy`
- `PersistentLoginStrategy`
- `PropertiesStrategy`
- `QueryStrategy`
- `RegistrationCodeStrategy`
- `RequestmapStrategy`
- `RoleStrategy`
- `UserStrategy`

The controllers, taglib, and even the service never call strategy methods directly on the service, only via a strategy interface.

Each interface has a default implementation, e.g. `DefaultAcLStrategy`, `DefaultErrorsStrategy`, etc., and these simply delegate to `SpringSecurityUiService` (except for `MailStrategy`, which has `MailPluginMailStrategy` as its default implementation which uses the Mail plugin to send emails). Each of the default implementations is registered as a Spring bean:

- `uiAcLStrategy`
- `uiErrorsStrategy`
- `uiMailStrategy`
- `uiPersistentLoginStrategy`
- `uiPropertiesStrategy`
- `uiQueryStrategy`
- `uiRegistrationCodeStrategy`
- `uiRequestmapStrategy`
- `uiRoleStrategy`

- `uiUserStrategy`

To override the functionality defined in one of the strategy interfaces, register your own implementation of the interface in your application's `grails-app/conf/spring/resources.groovy`, e.g.

```
import com.myapp.MyRequestmapStrategy

beans = {
    uiRequestmapStrategy(MyRequestmapStrategy)
}
```

and yours will be used instead.

11.6. Password Verification

By default the registration controller has rather strict requirements for valid passwords; they must be between 8 and 64 characters and must include at least one uppercase letter, at least one number, and at least one symbol from “!@#\$\$%^&”. You can customize these rules with these `application.groovy` attributes:

Property	Default Value
<code>grails.plugin.springsecurity.ui.password.minLength</code>	8
<code>grails.plugin.springsecurity.ui.password.maxLength</code>	64
<code>grails.plugin.springsecurity.ui.password.validationRegex</code>	<code>"^.*(?:.*\\d)(?=.*[a-zA-Z])(?=.*[!@#\$%^&]).*\$"</code>

Chapter 12. Scripts

12.1. s2ui-override

Purpose

Generates controllers that extend the plugin's controllers and copies their GSPs to your application for overriding of functionality.

The general format is:

```
grails s2ui-override <type> [controllerPackage]
```

The script will copy an empty controller that extends the corresponding plugin controller into your application so you can override individual actions and methods as needed. It also copies the controller's GSPs. The exceptions are when type is 'auth' or 'layout' which only copy GSPs.

See the [Customization](#) section for more details.

12.2. s2ui-create-challenge-questions

Purpose

Generates controllers, services, domain object, views, i18N, and application configuration to have challenge questions

The general format is:

```
grails s2ui-create-challenge-questions <domain-class-package> <challenge-qa-class-name> <user-domain-class-name> [number-of-questions]
```

- domain-class-package is required and is the package where the Domain, Controller, and Services will be created. It is recommend to use the same class where your 'User' class is created.
- challenge-qa-class-name is required and is the Name of the domain class used to store this information
- user-domain-class-name is required and is the package and name of the User class. It is recommend to ensure the first parameter is the same package as the User class.
- number-of-questions is optional but if not given will default to 2.

```
Example: s2ui-create-challenge-questions com.mycompany Profile com.mycompany.User 4
```

This script will create one Domain Class. This will contain a link to the user and also contain one entry for each questions and answer. If you pass in either no package or the same package as the what is used for the Challenge Question domain it will not put a package in front of the User class.

However, if you pass in a different package then it will add the package in front of the user class.

This script will create one controller which contains all the logic for handling the actions needed to provide a user interface to manage different users challenge questions and answers.

This script will create two services. The first service is to handle to be the implementation for the controller/domain object handling the data needed to set/get questions and answers. The second service is a listener service which ensures that anytime an answer is created or updated the value is encrypted before it is stores in the database.

This script will create three views. Each of the three views (List/index, edit, and create) will support all the CRUD operations.

This script will add lines into your application.groovy file in order to ensure the questions and answers are properly configured.

This script will insert one line into messages.properties. This will be the menu item in the nav bar for spring security.

```
spring.security.ui.menu.<domain-class-package>.<challenge-qa-class-name>=<challenge-qa-class-name> Questions"
```