|  |
| --- |
| Codi CMS |
| Descripció aspectes importants CMS |
|  |
|  |
| 09/04/2015 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Versió** | **Autor** | **Data** | **Revisió** | **Comentaris** |
| 1.0 | Jesús Campos Muñoz | 9-Abril -2015 |  | Inici documentació. |

**Índex**

[1 Introducció 7](#_Toc416697809)

[2 Canvis comuns (Editor i Visor) 8](#_Toc416697810)

[2.1 Incloure Bootstrap en una aplicació Grails 8](#_Toc416697811)

[2.1.1 Instal·lació de BootStrap 8](#_Toc416697812)

[2.1.2 Bootstrap: exemple senzill d’ús 10](#_Toc416697813)

[2.2 Incloure Spring Security CAS i Spring Security Core en una aplicació Grails 10](#_Toc416697814)

[2.2.1 Arxiu BuildConfig.groovy 10](#_Toc416697815)

[2.2.2 Modificacions al plugin 11](#_Toc416697816)

[2.2.3 Arxiu Config.groovy 12](#_Toc416697817)

[2.2.4 Dominis i controladors 14](#_Toc416697818)

[2.2.5 Com fer servir els plugins de seguretat 15](#_Toc416697819)

[2.3 Eliminar Spring Security CAS i Spring Security Core d'una aplicació Grails 15](#_Toc416697820)

[2.3.1 Arxiu Config.grrovy 15](#_Toc416697821)

[2.3.2 Arxiu BuildConfig.groovy 16](#_Toc416697822)

[2.3.3 Controladors 16](#_Toc416697823)

[2.4 Canviar el nom del servidor d’una aplicació Grails 17](#_Toc416697824)

[2.5 Arxius CSS i Javascript 17](#_Toc416697825)

[3 Aplicació Editor (gcrrhh) 19](#_Toc416697826)

[3.1 TreeTable 19](#_Toc416697827)

[3.1.1 Instal·lació del plugin TreeTable 19](#_Toc416697828)

[3.1.2 Ús del plugin TreeTable 20](#_Toc416697829)

[3.1.2.1 Exemple bàsic 20](#_Toc416697830)

[3.1.2.2 Exemple complex 22](#_Toc416697831)

[3.2 CKEditor 23](#_Toc416697832)

[3.2.1 Instal·lació i configuració del plugin CKEditor 24](#_Toc416697833)

[3.2.2 Arxiu de configuració config.js 25](#_Toc416697834)

[3.2.3 Resolució de problemes 28](#_Toc416697835)

[3.2.3.1 Generació de l’arxiu WAR 28](#_Toc416697836)

[3.2.3.2 CKEditor esborra les propietats i estils dels elements HTML 28](#_Toc416697837)

[4 Aplicació Visor (rrhh) 30](#_Toc416697838)

[5 Annexos 31](#_Toc416697839)

[5.1 Activar pestanya *Markers* 31](#_Toc416697840)

[5.2 Activar pestanya *Navigator* 31](#_Toc416697841)

[6 Referències 32](#_Toc416697842)

[6.1 Bootstrap 32](#_Toc416697843)

[6.2 Spring Security Core 32](#_Toc416697844)

**Imatges**

[Imatge 1: Esquelet aplicació Grails amb Bootstrap 9](#_Toc416690825)

[Imatge 2: Imatge de Bootstrap d'un exemple bàsic 10](#_Toc416690826)

[Imatge 3: Detecció de problemes del plugin Srping Security Core amb la pestanya Markers 11](#_Toc416690827)

[Imatge 4: Esquelet aplicació Grails amb TreeTable 19](#_Toc416690828)

[Imatge 5: Exemple sencill amb TreeTable. Vista 1 21](#_Toc416690829)

[Imatge 6: Exemple sencill amb TreeTable. Vista 2 21](#_Toc416690830)

[Imatge 7: Aparença del CKEditor amb configuració ràpida 25](#_Toc416690831)

**Figures**

[Figura 1: Referenciar Bootstrap en una aplicació Grails 9](#_Toc416690796)

[Figura 2: Codi d'exemple d'ús de Bootstrap 10](#_Toc416690797)

[Figura 3: Instal·lació del plugin Spring Security Core 11](#_Toc416690798)

[Figura 4: Instal·lació del plugin Spring Security CAS 11](#_Toc416690799)

[Figura 5: Ruta de ReflectionsUtils.groovy d'Spring Security Core a la pestanya Navigator 12](#_Toc416690800)

[Figura 6: Ruta de ReflectionsUtils.groovy d'Spring Security Core al sistema de fitxers 12](#_Toc416690801)

[Figura 7: Import erroni a ReflectionsUtils.groovy 12](#_Toc416690802)

[Figura 8: Import correcte a ReflectionsUtils.groovy 12](#_Toc416690803)

[Figura 9: Instrucció incorrecta a ReflectionsUtils.groovy 12](#_Toc416690804)

[Figura 10: Instrucció correcta a ReflectionsUtils.groovy 12](#_Toc416690805)

[Figura 11: Configuració en comú de seguretat dels diferents entorns 13](#_Toc416690806)

[Figura 12: Configuració de seguretat específica per entorns 13](#_Toc416690807)

[Figura 13: Nom del servidor d'exemple 13](#_Toc416690808)

[Figura 14: Possible error al configurar Spring Security 14](#_Toc416690809)

[Figura 15: *Config.grrovy* eliminar plugins de seguretat 15](#_Toc416690810)

[Figura 16: BuildConfig.grrovy eliminar plugins de seguretat 16](#_Toc416690811)

[Figura 17: aplication.properties eliminar plugins de seguretat 16](#_Toc416690812)

[Figura 18: Canviar el nom del servidor 17](#_Toc416690813)

[Figura 19: Canviar el nom del servidor a Grails 2.4.3 17](#_Toc416690814)

[Figura 20: Referenciar TreeTable en una aplicació Grails 20](#_Toc416690815)

[Figura 21: Codi exemple senzill amb TreeTable 21](#_Toc416690816)

[Figura 22: Crida al template que construeix el TreeTable 22](#_Toc416690817)

[Figura 23: Codi Javascript per executar TreeTable 23](#_Toc416690818)

[Figura 24: Instal·lació del plugin CKEditor 24](#_Toc416690819)

[Figura 25: Codi GSP per incloure CKEditor 24](#_Toc416690820)

[Figura 26: Tractar el text escrit al CKEditor 25](#_Toc416690821)

[Figura 27: Ruta arxiu de configuració config.js del CKEditor a la pestanya Navigator 25](#_Toc416690822)

[Figura 28: Ruta arxiu de configuració config.js del CKEditor al sistema de fitxers 25](#_Toc416690823)

[Figura 29: Ruta arxiu config.js al sistema de fitxers amb versions antigues de Grails 26](#_Toc416690824)

# Introducció

El gestor de continguts o CMS (Content Management System) dissenyat està format per dues aplicacions:

* L’aplicació ***gcrrhh*** o ***Editor*** (d’ara en endavant l’Editor): serveix per construir pàgines web basades en un determinat esquelet i decoració, que poden ser triats per l’usuari en funció del recursos disponibles. Aquestes pàgines es guarden en bases de dades.
* L’aplicació ***rrhh*** o ***Visor*** (d’ara en endavant el Visor): serveix per visualitzar les pàgines generades amb l’Editor.

Aquestes dues aplicacions estan programades fent servir el *framework* d’aplicacions web Grails i diferents plugins de la plataforma (com pot ser l’editor de text **CKEditor**) o *frameworks* addicionals (com és **Bootstrap**). Els plugins tenen unes determinades configuracions que cal tenir en compte per tal que es comportin com nosaltres volem, modificant el seu que tenen per defecte. A més, el *framworks* s’han d’instal·lar i configurar que poder utilitzar-los.

En les webs d’aquests complements s’explica com configurar-los, però a vegades es donen aspectes per soposats que no són tan directes de veure. En aquesta guia es pretén explicar com realitzar les instal·lacions i canvis adients per tal que tots el complements quedin configurats com volem, és a dir, com estan a l’actual CMS. En alguns casos fins i tot posant exemples senzills que es poden dur a terme en una aplicació Grails desde zero per tal de veure el seu comportament.

TAMBE ES PRETENEN DONAR DETALLS TECNICS (REALIZAR EL BUSCADOR, DETALLES DE CODIGO ETC)

# Canvis comuns (Editor i Visor)

Es detallen els canvis que s’han realitzat tant a l’Editor com al Visor. El framework de visualització per plataformes mòbils, Bootstrap, i els de seguretat, CAS i Spring Security Core es fan servir a ambdues aplicacions.

## Incloure Bootstrap en una aplicació Grails

Bootstrap és un *framework* que serveix per dissenyar aplicacions de movilitat. És l’encarregat, ajudant-se d’un conjunt d’arxius Javascript i CSS, d’adaptar els elements que de la pàgina web al dispositiu mòvil.

### Instal·lació de BootStrap

Descarregar el framework de la seva pàgina web: <http://getbootstrap.com/getting-started/#download>

És aconsellable descarregar la versió que es troba a : *”Bootstrap: Compiled and minified CSS, JavaScript, and fonts. No docs or original source files are included”*.

Només faran falta els següents arxius que es troben a la descompressió:

* css/bootstrap.css
* js/bootstrap.min.js
* La carpeta *fonts*, per poder fer servir el *glyphicons*.

Aquests, es copien en les següents ubicacions dintre del projecte Grails:

* web-app/css/bootstrap.css
* web-app/js/bootstrap.min.js
* web-app/fonts/

Per últim, s’han de referenciar els arxius JS i CSS a les vistes GSP. Generalment, el Bootstrap es farà servir a tota l’aplicació per la qual cosa la millor opció es fer les referències en una GSP que sigui un *layout*. Llavors al *layout* ***main.gsp*** s’ha d’insertar el següent codi:

|  |
| --- |
| <head>  ...  <link rel=*"apple-touch-icon"* sizes=*"114x114"* href=*"*${assetPath(src: 'apple-touch-icon-retina.png')}*"*>  <link rel=*"stylesheet"* href=*"/*${grails.util.Metadata.current.'app.name'}*/css/bootstrap.css"* type=*"text/css"*>  <script type=*"text/javascript"* src=*"/*${grails.util.Metadata.current.'app.name'}*/js/bootstrap.min.js"*></script>  <asset:stylesheet src=*"application.css"*/>  ...  </head> |

Figura 1: Referenciar Bootstrap en una aplicació Grails

A la següent imatge es pot veure com quedaria el projecte amb les carpetes corresponents a l’editor de codi GGTS:



Imatge 1: Esquelet aplicació Grails amb Bootstrap

### Bootstrap: exemple senzill d’ús

Per tal de comprovar que s’han referenciat de manera correcta els arxius del *framework* Bootstrap, el següent codi es pot incloure a una vista GSP:

|  |
| --- |
| <html>  <head>  <meta name=*"layout"* content=*"main"* />  <title>Proba Bootstrap</title>  </head>  <body>  <p>  <button type=*"button"* class=*"btn btn-lg btn-default"*>Default</button>  <button type=*"button"* class=*"btn btn-lg btn-primary"*>Primary</button>  <button type=*"button"* class=*"btn btn-lg btn-success"*>Success</button>  <button type=*"button"* class=*"btn btn-lg btn-info"*>Info</button>  <button type=*"button"* class=*"btn btn-lg btn-warning"*>Warning</button>  <button type=*"button"* class=*"btn btn-lg btn-danger"*>Danger</button>  <button type=*"button"* class=*"btn btn-lg btn-link"*>Link</button>  </p>  </body>  </html> |

Figura 2: Codi d'exemple d'ús de Bootstrap

I s’hauria de visualitzar una web molt similar a la següent:



Imatge 2: Imatge de Bootstrap d'un exemple bàsic

## Incloure Spring Security CAS i Spring Security Core en una aplicació Grails

Per instal·lar el CAS, que serveix per fer autenticar-se en diferents aplicacions només amb una única identificació, i l’Spring Security Core, que serveix per obtenir les credencials, es parteix de que alguns arxius com poden ser controladors o dominis s’extreuran d’altres aplicacions.

### Arxiu BuildConfig.groovy

Afegir en la closure **plugins** l’Spring Security Core:

|  |
| --- |
| plugins {  ...  compile ":spring-security-core:1.2.7.3"  ...  } |

Figura 3: Instal·lació del plugin Spring Security Core

Després, sobre el projecte fer botó dret → *Grails Tools* → *Refresh depencies* (Alt+G,R) per actualitzar les dependències del projecte.

Ara afegir a la closure **plugins** l’Spring Security CAS:

|  |
| --- |
| plugins {  ...  compile ":spring-security-core:1.2.7.3"  compile ':spring-security-cas:1.0.5'  ...  } |

Figura 4: Instal·lació del plugin Spring Security CAS

I tornar a refrescar dependències. Independentment de si s’obtenen errors o no a la a la consola del GGTS es reinicia l’entorn de desenvolupament.

### Modificacions al plugin

Un cop reiniciat l’entorn, si s’està fent servir la versió 2.4.3 de Grails o per sobre s’obtindrà un error al codi del plugin que s’ha descarregat.

És fàcil veure que hi ha un problema al projecte perquè apareix una X vermella, peró no ho és tant detectar on.

Una opció és mirar a la pestanya de *Markers* (si no està activada consultar **5.1 Activar pestanya *Markers***) on s’ha produït l’error, com es mostra a la següent imatge:



Imatge 3: Detecció de problemes del plugin Srping Security Core amb la pestanya Markers

A la pestanya *Markers* s’ens mostra l’arxiu conflictiu, indicant la seva ruta per poder localitzar-lo i en quina aplicació està causant el problema. Peró de manera molt senzilla, fent doble click sobre *Groovy unable to resolve...*, d’aquesta manera s’obrirà a l’editor l’arxiu problemàtic.

Una altra és anar directament a l’arxiu en concret ja que és un error detectat (arxiu ***ReflectionsUtils.groovy***) entre la versió de Grails 2.4.3 i superiors i Spring Security Core. La ruta de l’arxiu és:

|  |
| --- |
| /gcrrhh/.link\_to\_grails\_plugins/spring-security-core-1.2.7.3/src/groovy/org/codehaus/groovy/grails/plugins/springsecurity/ReflectionUtils.groovy |

Figura 5: Ruta de ReflectionsUtils.groovy d'Spring Security Core a la pestanya Navigator

Per accedir desde l’entorn de desenvolupament s’ha de fer a través de la pestanya *Navigator* (si no està activada consultar **5.2 Activar pestanya *Navigator***).

Per accedir desde el sistema de fitxers:

|  |
| --- |
| dir\_workspace\dir\_aplicacio\target\work\plugins\spring-security-core-1.2.7.3\src\groovy\org\codehaus\groovy\grails\plugins\springsecurity\ReflectionUtils.groovy |

Figura 6: Ruta de ReflectionsUtils.groovy d'Spring Security Core al sistema de fitxers

Llavors, dintre de l’arxiu ReflectionsUtils substituir:

|  |
| --- |
| import org.codehaus.groovy.grails.commons.ApplicationHolder |

Figura 7: Import erroni a ReflectionsUtils.groovy

per:

|  |
| --- |
| import grails.util.Holders |

Figura 8: Import correcte a ReflectionsUtils.groovy

Substituir:

|  |
| --- |
| application = ApplicationHolder.application |

Figura 9: Instrucció incorrecta a ReflectionsUtils.groovy

per:

|  |
| --- |
| application = Holders.grailsApplication |

Figura 10: Instrucció correcta a ReflectionsUtils.groovy

### Arxiu Config.groovy

Al final d’aquest arxiu s’han d’incloure les següents línies:

|  |
| --- |
| // Added by the Spring Security Core plugin:  grails.plugins.springsecurity.userLookup.userDomainClassName='Security.SecUser'  grails.plugins.springsecurity.userLookup.authorityJoinClassName= 'Security.SecUserSecRole'  grails.plugins.springsecurity.authority.className = 'Security.SecRole'  grails.plugins.springsecurity.securityConfigType = 'Annotation'  grails.plugins.springsecurity.rejectIfNoRule = **false**  grails.plugins.springsecurity.cas.active = **true**  grails.plugins.springsecurity.cas.sendRenew = **false**  grails.plugins.springsecurity.cas.serverUrlEncoding = 'UTF-8'  grails.plugins.springsecurity.cas.key = 'grails-spring-security-cas'  grails.plugins.springsecurity.cas.artifactParameter = 'ticket'  grails.plugins.springsecurity.cas.serviceParameter = 'service'  grails.plugins.springsecurity.cas.filterProcessesUrl = '/j\_spring\_cas\_security\_check' |

Figura 11: Configuració en comú de seguretat dels diferents entorns

Després, per a cada entorn (dev, test i prod) s’han de configurar les següents línies:

|  |
| --- |
| environments {  ...  dev { //Per exemple  grails.plugins.springsecurity.cas.loginUri = '/login'  grails.plugins.springsecurity.cas.serviceUrl = "http://localhost:8099/${grails.util.Metadata.current.'app.name'}/j\_spring\_cas\_security\_check"  grails.plugins.springsecurity.cas.serverUrlPrefix = 'https://cas.upc.edu/'  grails.plugins.springsecurity.cas.proxyCallbackUrl = "http://localhost:8099/${grails.util.Metadata.current.'app.name'}/secure/receptor"  grails.plugins.springsecurity.cas.proxyReceptorUrl = '/secure/receptor'  grails.plugins.springsecurity.cas.useSingleSignOut = 'true'  grails.plugins.springsecurity.logout.afterLogoutUrl = "https://cas.upc.edu/logout?url=http://localhost:8099/${grails.util.Metadata.current.'app.name'}"  ...  }  ...  } |

Figura 12: Configuració de seguretat específica per entorns

Per fer-ho, s’ha suposat que a desenvolupament es té la següent configuració:

|  |
| --- |
| environments {  ...  dev {  ...  grails.serverURL = http://localhost:8099/${grails.util.Metadata.current.'app.name'}  ...  }  ...  } |

Figura 13: Nom del servidor d'exemple

On /${grails.util.Metadata.current.'app.name'} indica el nom de l’aplicació de manera genèrica en comptes de posar *rrhh* o *gcrrhh*. Aixó és útil per quan una aplicació no te un nom definitiu.

S’ha d’anar amb compte l’entorn que e configura, ja que habitualment els entorns de test i producció acostumen a tenir una connexió de tipus https, mentres que a desenvolupament és http. Es podria donar el següent cas (suposant la configuració de la Figura 13: Nom del servidor d'exemple):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | grails.plugins.springsecurity.cas.loginUri = '/login'  grails.plugins.springsecurity.cas.serviceUrl =  "https://localhost:8090/${grails.util.Metadata.current.'app.name'}/j\_spring\_cas\_security\_c  heck"  grails.plugins.springsecurity.cas.serverUrlPrefix = 'https://cas.upc.edu/'  grails.plugins.springsecurity.cas.proxyCallbackUrl =  "https://localhost:8090/${grails.util.Metadata.current.'app.name'}/secure/receptor"  grails.plugins.springsecurity.cas.proxyReceptorUrl = '/secure/receptor'  grails.plugins.springsecurity.cas.useSingleSignOut = 'true'  grails.plugins.springsecurity.logout.afterLogoutUrl  ="https://cas.upc.edu/logout?url=https://localhost:8090/${grails.util.Metadata.current.'app  .name'}" |

Figura 14: Possible error al configurar Spring Security

A les línies 2-3, 6-7 i 10-11 hi han erros. Estan intentant accedir mitjançant https quan la configuració és http.

### Dominis i controladors

Ara s’haurien de crear els dominis del package **Security** (*SecRole, SecUser,* *SecUserSecRole*) i els controladors del package **Security** (*Login, Logout*).

Llavors, per no haver de crear i configurar desde zero els dominis i els controlador es copien d’altre projecte tenint cura de el package Security quedi com al projecte del que es copien.

Una vegada s’han copiat els dominis cal modificar a la closure *static mapping* la variable *table* que hi ha dintre de cada un d’ells per tal d’apuntar a les taules correctes.

Com que la inclusió de determinades parts de la seguretat es fa per herència en altres aplicacions, extenent de la classe ***BaseCtronler.groovy***, la solució és copiar els arxius necessaris d’altres aplicacions com pugin ser Tempus (RLG), Salut Cardiovascular (SAL), etc ...

Primer de tot s’agafa el package **utils** (src/groovy/utils) d’altre projecte i es copia al nostre en la mateixa ubicació. Aquest package té cinc arxiu: **BaseControler*.groovy*, Crypt*.groovy*, Database*.groovy*, FileUploader*.groovy*, Validation*.groovy***.

### Com fer servir els plugins de seguretat

Fins ara s’ha explicat quins són els plugin necessaris per tenir seguretat a la nostra aplicació i com configurar-los.

Ara s’explicarà com fer-los servir per tal de prohibir la visualització de vistes i/o controladors a determinats perfils.

Per tal que un controlador tingui en compte el rol de l’usuari:

* Ha d’extendre/heretar de ***BaseControler.groovy***
* Ha d’importar:
  + utils.BaseControler
  + grails.plugins.springsecurity.Secured
* Abans del nom de la classe ha de tenir la sentència: @Secured(['ROLE']), on ‘ROLE’ ha de ser el rol d’usuari a vigilar (ROLE\_ADMIN, ROLE\_USER, etc...)
* En l’acció en que accedim per primera vegada al controlador s’ha d’executar la instrucció: this.populateUserName().

## Eliminar Spring Security CAS i Spring Security Core d'una aplicació Grails

Una vegada l’aplicació està ben configurada amb els plugins de seguretat pot ser necessari per segons quin tipus de proves eliminar-los, com per exemple a les proves de càrrega o estrés per tal d’accedir-hi sense indentificar-se.

Com que desfer tot els passos de l’apartat anterior (**2.2 Incloure Spring Security CAS i Spring Security Core en una aplicació Grails**) pot ser una mica tediós i no totes les aplicacion estan fetes de la mateixa manera, en aquest apartat es descriu d’una manera ràpida i genèrica.

### Arxiu Config.grrovy

En els respectius entorns comentar totes les línies que comencin per: *grails.plugins.springsecurity.cas*.

Comentar les línies del final de l’arxiu:

|  |
| --- |
| //// Added by the Spring Security Core plugin:  //grails.plugins.springsecurity.userLookup.userDomainClassName = 'Security.SecUser'  //grails.plugins.springsecurity.userLookup.authorityJoinClassName = 'Security.SecUserSecRole'  //grails.plugins.springsecurity.authority.className = 'Security.SecRole'  //grails.plugins.springsecurity.securityConfigType = 'Annotation'  //grails.plugins.springsecurity.rejectIfNoRule = false |

Figura 15: *Config.grrovy* eliminar plugins de seguretat

### Arxiu BuildConfig.groovy

Comentar a la closure **plugins** les següents línies:

|  |
| --- |
| plugins {  ...  // compile ":spring-security-core:1.2.7.3"  // compile ':spring-security-cas:1.0.5'  ...  } |

Figura 16: BuildConfig.grrovy eliminar plugins de seguretat

En el cas d’aplicacions realitzades amb la versión de Grails 2.1.1 s’ha de desinstal·lar mitjançant el **Plugin Manager** (botó dret sobre plugins a l’arbre del projecte en el GGTS, *Open Grails Plugin Manager*, seleccionar el plugin i desinstlar-ho) o bé comentar les següents línies a l’arxiu ***aplications.properties***:

|  |
| --- |
| #Grails Metadata file  #Wed Jan 22 12:22:56 CET 2014  app.grails.version=2.1.1  ...  #plugins.spring-security-cas=1.0.5  #plugins.spring-security-core=1.2.7.3 |

Figura 17: aplication.properties eliminar plugins de seguretat

### Controladors

S’ha d’eliminar la següent sentència en tots el controladors on estigui declarada: @Secured(['ROLE']), on ‘ROLE’ és el rol d’usuari a vigilar (ROLE\_ADMIN, ROLE\_USER, etc...).

Eliminar import grails.plugins.springsecurity.Secured.

A partir d’ara hi ha dues opcions:

* 1ª opció: Evitar que els controladors heretin de ***BaseControler.groovy*** i eliminar les sentències:
  + this.populateUserName()
  + this.doSecurityCheck()
* 2ª opció: Modificar ***BaseControler.groovy*** per tal de no fer més modificacions als controladors. S’ha de fer un hardcode de les dades degut a que es necessita el PERNR de l’usuari i el seu nom per fer-lo servir (s’han de fer servir uns que ja existeixin a la base de dades). A /src/groovy/utils/BaseControler.groovy s’han de canviar els següents métodes:
  + def populateUserName() queda de la següent manera:

|  |
| --- |
| def populateUserName()  {  session["user"] = "jesus.campos"  } |

* + Afegir a l’inici del doSecurityCheck():

session.user="jesus.campos"

## Canviar el nom del servidor d’una aplicació Grails

Habitualment per canviar el nom del servidor de *localhost* a *pc24094.upc.es*, per exemple, s’ha de modificar la closure ***environments*** de l’entorn corresponent a l’arxiu **Config.groovy** de la següent manera:

|  |
| --- |
| environments {  ...  test {  ...  grails.serverURL = "http://pc24094.upc.es:8099/ ${grails.util.Metadata.current.'app.name'}"  ...  }  ...  } |

Figura 18: Canviar el nom del servidor

En el cas de Grails 2.4.3 aixó no és suficient, a més a l’arxiu **BuildConfig.groovy** s’ha de comentar les egüents línies dintre de la closure ***grails.project.fork***:

|  |
| --- |
| grails.project.fork = [  ...  // configure settings for the run-app JVM  // run: [maxMemory: 768, minMemory: 64, debug: false, maxPerm: 256, forkReserve:false],  ...  ] |

Figura 19: Canviar el nom del servidor a Grails 2.4.3

## Arxius CSS i Javascript

Tant pel que fa a l’Editor com al Visor es fan servir diferents arxius CSS, alguns dels qual son personalitzacions dels CSS del *framework* Bootstrap. També es fan servir arxius Javascript.

Als següents arxius compressos hi ha un recull dels més importants, inclosa la configuració del plugin Ckeditor.



# Aplicació Editor (gcrrhh)

El nom genèric de l’aplicació al servidor és gcrrhh i és coneguda com a l’Editor. Serveix per construir pàgines web amb un determinat format aplicacble per part de l’usuari, els Templates. Els Templates serveixen per modifcar el marc de la pàgina, és a dir, capceleres i peus de pàgina de pàgina que es veuran, etc. També existeixen un formats anomenats Plantilles, els quals faciliten la tasca de la creació del contingut i permeten seguir un determinat patró de disseny al temps que estalvia a l’editor de contingut (referint-se al dissenyador) el codi CSS i JS per realitzar element amb moviment (desplegables).

## TreeTable

Plugin per visualitzar el contingut de l’Editor de manera ordenada.

### Instal·lació del plugin TreeTable

Descarregar l’arxiu comprimit (opció **“Download Now”**) del següent enllaç: <https://plugins.jquery.com/treetable/>

Només faran falta els següents arxius que es troben a la descompressió:

* El arxius continguts a la carpeta css.
* L’arxiu jquery.treetable.js de l’arrel.

Aquests, es copien en les següents ubicacions dintre del projecte Grails:

* El arxius continguts a la carpeta css a la carpeta web-app/css/treeTable
* web-app/css/treeTable/jquery.treetable.js

A la següent imatge es pot veure com quedaria el projecte amb les carpetes corresponents a l’editor de codi GGTS:



Imatge 4: Esquelet aplicació Grails amb TreeTable

Per últim, s’han de referenciar els arxius JS i CSS a les vistes GSP. Generalment, el TreeTable es farà servir només per visualitzar tots els continguts que es poden editar, peró no quan s’estigui editant una pàgina. Per referenciar el plugin TreeTable en un la un *layout* (a l’aplicació es troba al ***main.gsp***) s’ha d’insertar el següent codi:

|  |
| --- |
| <head>  ...  <meta name=*"layout"* content=*"main"* />  <link href=*"/*${grails.util.Metadata.current.'app.name'}*/css/treeTable/jquery.treetable.css"* rel=*"stylesheet"* type=*"text/css"* />  <link href=*"/*${grails.util.Metadata.current.'app.name'}*/css/treeTable/jquery.treetable.theme.default.css"* rel=*"stylesheet"* type=*"text/css"* />  <script src=*"/*${grails.util.Metadata.current.'app.name'}*/js/treeTable/jquery.treetable.js"*>  </script>  ... |

Figura 20: Referenciar TreeTable en una aplicació Grails

### Ús del plugin TreeTable

Per mostrar el comportament del plugin es mostren dos exemples:

* El bàsic és per veure simplement el seu funcionament.
* L’exemple complex serveix per veure com està implementat realment en l’aplicació de l’Editor. Aquest descriu detalladament la configuració a l’aplicació.

#### Exemple bàsic

Per veure la funcionalitat del plugin només cal copiar el següent codi a una pàgina GSP que tingui les referències de l’apartat anterior:

|  |
| --- |
| <table id="example-basic-expandable">  <tr class="blank" data-tt-id="blank00">  <td>  <span class="blank">File 1</span>  </td>  <td>File #1</td>  <td></td>  </tr>  <tr class="branch" data-tt-id="00">  <td>  <span class="file">File 2</span>  </td>  <td>File #2</td>  <td></td>  </tr>  <tr class="branch" data-tt-id="01">  <td>  <span class="folder">Folder 1</span>  </td><td>Folder #1</td>  <td>--</td>  </tr>  <tr class="branch" data-tt-id="02" data-tt-parent-id="01">  <td>  <span class="folder">Folder 2</span>  </td><td>Folder #2</td>  <td>--</td>  </tr>  <tr class="branch" data-tt-id="03" data-tt-parent-id="02">  <td>  <span class="folder">Folder 3</span>  </td><td>Folder #3</td>  <td>--</td>  </tr>  <tr class="branch" data-tt-id="04" data-tt-parent-id="02">  <td>  <span class="file">File 3</span>  </td><td>File #3</td>  <td>--</td>  </tr>  </table>  <script>  $("#example-basic-expandable").treetable({ expandable: **true** });  </script> |

Figura 21: Codi exemple senzill amb TreeTable

A l’executar el codi anterior s’obté al navegador web:



Imatge 5: Exemple sencill amb TreeTable. Vista 1

Expandint el node *Folder 1* que té fills:



Imatge 6: Exemple sencill amb TreeTable. Vista 2

A partir d’ara es descriu el codi. Primer de tot cal dir que el plugin es basa en incloure-ho tot en una taula. L’atribut data-tt-id de cada tag *tr* serveix d’atribut id, és a dir, ha de ser únic. Aquest atribut és útil en el cas de voler fer que un element contingui d’altres en el seu interior. Com pot ser el cas del *Folder 2*, que té dintre *Folder 3* i *File 3*, llavors *Folder 2* seria l’element pare i elas altres els fills. Per exemple, si es vol que un element estigui dintre d’altre, l’element fill ha de tenir l’atribut data-tt-parent-id on s’indicarà el valor de l’atribut data-tt-id de l’element pare. En aquest exemple el valor data-tt-parent-id del elements és *02*, que coincideix amb el valor de data-tt-id del *Folder 2*.

En aquest cas la fila es divideix en tres columnes, aixó pot variar. La imatge del triangle indicant que l’element té fills apareix automàticament en el cas de que el valor de l’atribut data-tt-id de l’element apareixi en l’atribut data-tt-parent-id d’un altre element. Les imatges de carpetes i fitxers es poden modificar amb l’atribut class del tag *span* on estan ubicades, indicant *folder* o bé *file* com a valor.

Per finalitzar, per tal d’indicar en quin element es vol aplicar el plugin TreeTable s’ha de posar el codi Javascript del final. S’ha d’executar després de que es carregui la pàgina. És similar a l’ús d’altres plugins, com podria ser el de incloure un calendari, i permet personalitzar-lo de manera similar, fent servir aquesta funció Javascript.

Per a més informació sobre la personalització del plugin: <http://ludo.cubicphuse.nl/jquery-treetable/>

#### Exemple complex

Aquest exemple es basa en fer servir un template de Grails creat expressament per mostrar les files.

|  |
| --- |
| <table id="expandable-tree-table-plantillas">  <thead>  <tr>  <th width="90%">Plantilla</th>  <th width="10%">Accions</th>  </tr>  </thead>  <tbody>  <g:each in="${plantillasContent}" var="c">  <g:render template="newTreeTableNode" model="[c:c, tipDoc:'plantilles']"/>  </g:each>  </tbody>  </table> |

Figura 22: Crida al template que construeix el TreeTable

El template no és més que una vista GSP amb el caràcter “\_” com a primera lletra del seu nom. D’aquesta manera s’aconsegueix carregar el codi d’una vista GSP dintre d’una altra GSP.

Una vegada es tenen les dades del controlador, cada element tindrà com a atribut data-tt-id el seu id de la base de dades, i en el cas que tingui fills, cridarà novamente al template que s’encarregarà de construir una nova fila per cada fill, on cada fill tindrà un data-tt-parent-id corresponent a l’identificador del pare que té indicat a la base de dades. Aquest comportament és recursiu, llavors si un fill té fills no hi ha problema, no és processarà el següent fill fins que no s’hagin tractat tots els descendents de l’element actual.

En aquesta ocasió la crida Javascript es realitza al principi de la GSP fent servir el codi de la línia 2 que serveix per executar el codi Javascript quan s’ha carregat tota la pàgina.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <script type="text/javascript">  $(document).ready(**function**(){  //TreeTable JQuery Plugin  // Highlight selected row  $("#expandable-tree-table-templates tbody, #expandable-tree-table-  plantillas tbody").on("mousedown", "tr", **function**() {  $(".selected").not(**this**).removeClass("selected");  $(**this**).toggleClass("selected");  });  $("#expandable-tree-table-templates, #expandable-tree-table-  plantillas").treetable({ expandable: **true** });  });  </script> |

Figura 23: Codi Javascript per executar TreeTable

El codi contingut entre les línies 5 i 9 serveix per il·luminar en un color diferent la columna sobre la que s’ha efectuat un click. Permet seleccionar i des-seleccionar una fila.

## CKEditor

Plugin que serveix per editar el contingut en mode [WYSIWYG](http://ca.wikipedia.org/wiki/WYSIWYG), és a dir, el que és veu a la pantalla d’edició és el que veurà l’usuari a la web final. És conegut també com a *rich editor*, donat que no és un editor de text pla i sense format.

Tots els exemples i explicacions estan provats amb la versió 4.4.1.0. La versió 3.6.6.1.0 (que és la que fa servir el gestor de continguts Weceem) funciona pràcticament igual excepte en dues situacions:

* En el moment de crear un botó personalitzat amb funcions própies.
* En el moment de enviar les dades al controlador: la versió 3.6.6.1.0 envia el codi HTML amb indentacions mentres que la 4.4.1.0 no ho fa, s’ha de configurar expressament.

Aquest plugin té unes certes limitacions. Una és que s’ha de configurar de manera adequda, la configuració per defecte no serveix, per tal que no esborri els estils o atributs de determinats tags HTML (aixó es resol amb XXX). La configuració per defecte esborra o no té en compte els següents tags HTML: *<html>, <head> i <body>* (aixó es resol amb XXX).

### Instal·lació i configuració del plugin CKEditor

En l’arxiu **BuildConfig.groovy** s’ha d’afegir la següent línea:

|  |
| --- |
| grails.project.dependency.resolution = {  ...  plugins {  ...  compile ":ckeditor:4.4.1.0"  ...  }  } |

Figura 24: Instal·lació del plugin CKEditor

Després, sobre el projecte fer botó dret → *Grails Tools* → *Refresh depencies* (Alt+G,R) per actualitzar les dependències del projecte.

Amb aixó, i el següent codi a una vista GSP:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <head>  <ckeditor:resources/>  </head>  <body>  <form>  <ckeditor:editor name=*"text"* height=*"400px"* width=*"80%"*>${initialValue}</ckeditor:editor>  <g:actionSubmit action=*"save"* value=*"Desar"* />  </form>  </body> |

Figura 25: Codi GSP per incloure CKEditor

On ${initialValue} és el text que apareixerà d’inici a l’editor. Per tractar el text que s’ha escrit a l’editor és necessari recuperar la variable params.text. Llavors el controlador que crida a l’editor ha de tenir la següent acció:

|  |
| --- |
| **def** index() {  [initialValue: "Text inicial a mostrar"]  }    **def** save() {  *println* "Text: \n"+params.text  redirect(action: "index")  } |

Figura 26: Tractar el text escrit al CKEditor

Es pot veure l’aparença de l’editor, que hauria de semblar-se a la de la imatge de sota:



Imatge 7: Aparença del CKEditor amb configuració ràpida

### Arxiu de configuració config.js

Si es volen editar determinades propietats o modificar alguns comportaments de l’editor es pot fer mitjançant l’arxiu de configuració del plugin anomenat config.js. Aquest arxiu es troba al plugin, llavors la forma d’accedir a ell és a través de la ruta:

|  |
| --- |
| /nom\_aplicacio/.link\_to\_grails\_plugins/ckeditor-4.4.1.0/web-app/js/ckeditor/config.js. |

Figura 27: Ruta arxiu de configuració config.js del CKEditor a la pestanya Navigator

Per accedir desde l’entorn de desenvolupament s’ha de fer a través de la pestanya *Navigator* (si no està activada consultar **5.2 Activar pestanya *Navigator***).

Si es vol accedir a desde el sistema de fitxers:

|  |
| --- |
| dir\_workspace\dir\_aplicacio\target\work\plugins\ckeditor-4.4.1.0\web-app\js\ckeditor\config.js |

Figura 28: Ruta arxiu de configuració *config.js* del CKEditor al sistema de fitxers

En versions anteriors a Grails 2.4.3 aquesta carpeta en el sistema de fitxers en lloc de trobar-se al propi *workspace* es trobaria ubicada en la carpeta del usuari en una ruta similar a:

|  |
| --- |
| C:\Users\Jesus Campos\.grails\2.3.7\projects\nom\_aplicacio\plugins\ckeditor-4.4.1.0\web-app\js\ckeditor\config.js |

Figura 29: Ruta arxiu *config.js* al sistema de fitxers amb versions antigues de Grails

Si obrim aquest arxiu veurem que té el següent codi:

|  |
| --- |
| /\*\*  \* **@license** Copyright (c) 2003-2013, CKSource - Frederico Knabben. All rights reserved.  \* For licensing, see LICENSE.html or http://ckeditor.com/license  \*/  CKEDITOR.editorConfig = **function**( config ) {  // Define changes to default configuration here. For example:  // config.language = 'fr';  // config.uiColor = '#AADC6E';  }; |

Figura 30: Arxiu *config.js* original

Per fer-nos una idea del que es pot arribar a personalitzar l’editor podem posar el següent codi:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | CKEDITOR.editorConfig = **function**( config ) {  // Define changes to default configuration here. For example:  config.language = 'ca';  config.uiColor = '#69FFCD';    config.toolbar = [  { name: 'document', items: [ 'Image','Source', '-', 'NewPage', 'Preview'  ]}, { name: 'clipboard', items: [ 'Cut', 'Copy', 'Paste', 'PasteText',  'PasteFromWord', '-', 'Undo', 'Redo' ] },  '/',  { name: 'basicstyles', items: [ 'Bold', 'Italic' ] }  ];  }; |

Figura 31: Arxiu *config.js* modificat

Amb l’anterior exemple es pot veure com l’editor ja no té el matiex nombre de botons per a l’edició de text, degut a a que entre les línies 6 i 12 s’ha personalitzat la barra d’eines amb config.toolbar = [...] . El color del marc on tenen cabuda els botons s’ha modificat a la línia 4, i es poden fer servir valor RGB, hexadecimal o el nom del color com per exemple: red, blue, etc. A la línia 3 s’ha configurat el llenguatge dels *tooltips* d’ajuda per a cada botó.

En la imatge següent es pot veure com quedaria l’editor:



Figura 32: Aparença del CKEditor amb l’arxiu *config.js* modificat

Ara bé, per norma general els plugins no es pugen als repositoris i com s’ha vist, en versions antigues l’arxiu ***config.js*** es troba fora del *workspace*. Tot aixó fa que aquest arxiu sigui fàcil de perdre, la qual cosa ens indica que potser hi ha un altre lloc on posar tota aquesta configuració de manera fàcil i que estigui segura de possibles pèrdues. A més si la nova ubicació pot ser igual d’accessible que la dels arxius de configuració, controladors o vistes, molt millor. Tot aixó existeix, i es pot aconseguir afegint a l’arxiu ***Config.groovy*** de l’aplicació principal el següent codi:

|  |
| --- |
| ckeditor {  ...  config = "/js/myckconfig.js"  ...  } |

Figura 33: Reubicació de l'arxiu *config.js*

Amb aquesta configuració, l’arxiu ***config.js***s’ubicarà a la carpeta */web-app/js* del projecte principal i l’arxiu s’anomenarà ***myckconfig.js*** (pot ser un altre nom). Llavors tots els canvis que vulguem fer s’escriuran aquí i no es tindran en compte els que hi hagin dintre del que es torba a la ruta del plugin.

A partir d’ara, es parlarà d’arxiu de configuració o arxiu ***config.js*** per referir-se a ambdos arxius, ***config.js*** del plugin i ***myckconfig.js*** (o el seu equivalent triat pel lector). Quan s’indiqui que una línia de codi va situada en aquest arxiu, cadascú sabrà on posar-la.

L’arxiu de configuració que es fa servir a l’aplicació Editor (gcrrhh) és:

### Arxiu Config.groovy

A més de l’arxiu ***config.js***, a l’arxiu ***Config.groovy*** de l’aplicació principal també es poden configurar determinats elements, com s’ha vist amb l’arxiu ***config.js*** a l’apartat anterior.

Aquest tipus de configuracions estan relacionades amb el tipu de fitxers que es poden pujar, la seva ubicació, etc...

El codi que L’arxiu de configuració que es fa servir a l’aplicació Editor (gcrrhh) és:

|  |
| --- |
| ckeditor {  config = "/js/ckeditor/ckeditorconfig.js"  //skipAllowedItemsCheck = false  defaultFileBrowser = "ofm"  upload {  basedir = "/app/rrhh\_resources/"  baseurl ="/${grails.util.Metadata.*current*.'app.name'}*/../rrhh\_resources/"*  overwrite = **false**  link {  browser = **true**  upload = **false**  allowed = ['pdf', 'doc', 'docx']  denied = []  }  image {  browser = **true**  upload = **true**  allowed = ['jpg', 'gif', 'jpeg', 'png']  denied = []  }  flash {  browser = **false**  upload = **false**  allowed = ['swf']  denied = []  }  }  } |

### Resolució de problemes

Els següents subapartats són configuracions per personalitzar el plugin CKEditor com ho són els anteriors, peró aquestes s’han de realitzar per corretgir errors del comportament del plugin per tal que l’aplicació pugui funcionar. Potser amb altres versions de Grails a la indicada amb que succeeix el problema o altra versió del CKEditor no es donen.

#### Generació de l’arxiu WAR

Una aplicació que estigui realitzada amb Grails 2.4.3 i incorpori el plugin CKEditor 4.4.1.0 pot donar problemes a l’hora de realitzar l’arxiu WAR. Aixó és degut a que un dels plugins que ve per defecte amb la creació d’un projecte amb la versió de Grails esmentada és asset-pipeline:1.9.6. Aquest plugin s’encarrega de minimitzar el tamany de l’aplicació en el moment de realitzar el WAR, comprimint arxius CSS i JS, entre d’altres. El problema és que aquest plugin no pot comprimir el plugin CKEditor i l’error no permet construïr l’arxiu WAR.

La solució és comentar a l’arxiu ***BuildConfig.groovy*** la línia on s’afegeix aquest plugin:

|  |
| --- |
| plugins {  ...  //compile ":asset-pipeline:1.9.6"  ...  } |

#### CKEditor esborra les propietats i estils dels elements HTML

Si al *rich editor* ens arriba el següent codi/text:

|  |
| --- |
| <p class="txt-descripcio">Informaci&oacute; relacionada amb  l&#39;acc&eacute;s a la Universitat (acollida, declaraci&oacute;  d&#39;activitats),  classificaci&oacute; dels col&middot;lectius PAS i PR, Relaci&oacute;  de Llocs de Treball (descripci&oacute; i perfils) i  desvinculaci&oacute; (ren&uacute;ncia).</p> |

Quan la vista GSP enviï aquestes dades cap al controlador, el CKEditor amb la vista en *rich editor* i no en *Codi font* haurà esborrat class="txt-descripcio" i es perdrà per sempre. Llavors, que podem fer per solucionar-ho?

Incloure a l’arxiu ***config.js*** (o el seu equivalent) el següent codi:

|  |  |
| --- | --- |
| 1  2  3  4 | CKEDITOR.editorConfig = **function**( config ) {  config.allowedContent=true  config.extraAllowedContent = '\*(\*)';  }; |

Referència: <http://stackoverflow.com/questions/15659390/ckeditor-automatically-strips-classes-from-div>

# Aplicació Visor (rrhh)

# Annexos

## Activar pestanya *Markers*

Si la pestanya *Markers* no està activada cal anar a *Window* → *Show View* → *Other...* i escriure *Markers*, acte seguit apareixerà la paraula *Markers* a sota de la carpeta *General.*

## Activar pestanya *Navigator*

Si no està activada cal anar a *Window* → *Show View* → *Other...* i escriure *Navigator*, acte seguit apareixerà la paraula *Navigator* a sota de la carpeta *General.*

# Referències

## Bootstrap

Manual de Bootstrap: <https://librosweb.es/libro/bootstrap_3/>

## Spring Security Core

Explicació de com crear els dominis i els controladors que a l’apartat **2.2 Incloure Spring Security CAS i Spring Security Core en una aplicació Grails** s’han suggerit que era millor copiar-los d’altres aplicacions: <http://spring.io/blog/2010/08/11/simplified-spring-security-with-grails/>