



Eusko Jaurlaritzaren
Informatika Elkarte

Sociedad Informática
del Gobierno Vasco

Guía de uso: Integración UDA - Hdiv

UDA – Utilidades de Desarrollo de Aplicaciones

Este documento es propiedad de Eusko Jaurlaritzaren Informatika Elkartea – Sociedad Informática del Gobierno Vasco, S.A. (EJIE) y su contenido es CONFIDENCIAL. Este documento no puede ser reproducido, en su totalidad o parcialmente, ni mostrado a otros, ni utilizado para otros propósitos que los que han originado su entrega, sin el previo permiso escrito de EJIE. En el caso de ser entregado en virtud de un contrato, su utilización estará limitada a lo expresamente autorizado en dicho contrato. EJIE no podrá ser considerada responsable de eventuales errores u omisiones en la edición del documento.

| Versión | Fecha | Resumen de cambios | Elaborado por: | Aprobado por: |
|---------|------------|---|----------------|---------------|
| 1.0.0 | 07/06/2021 | Primera versión | Xabier Aldama | |
| 1.1.0 | 17/12/2021 | Añadir información acerca de los mensajes de error | Hugo Martínez | |
| 1.2.0 | 21/12/2021 | Añadir información sobre cambios importantes, explicación sobre cómo generar una aplicación a través del plugin de UDA y correcciones en general. | Xabier Agustín | |
| 1.3.0 | 20/06/2022 | Eliminar apartado de generación de aplicación por haber sido migrado al documento Plugin_UDA-Guia_de_uso_del_plugin.pdf | Xabier Agustín | |
| | | | | |

Índice

| | | |
|----------|--|-----------|
| 1 | Introducción | 4 |
| 2 | Validaciones..... | 5 |
| 2.1 | Control de acceso..... | 5 |
| 2.2 | Validación de editables | 5 |
| 2.3 | Validación de integridad | 5 |
| 3 | Instalación de la librería..... | 6 |
| 3.1 | Importación de dependencias..... | 6 |
| 3.2 | Configurar filtro y listener | 7 |
| 3.3 | Configurar clase de parametrización..... | 7 |
| 4 | Mecanismos de protección | 10 |
| 4.1 | Habilitar acceso..... | 10 |
| 4.2 | Envío de datos..... | 12 |
| 4.3 | Path variables | 15 |
| 5 | Configuración | 16 |
| 5.1 | Url exclusions - Start Pages..... | 16 |
| 5.2 | Text fields validation | 16 |
| 5.3 | Parameter exclusions - Start Parameters (NO recomendado) | 17 |
| 5.4 | Parámetros generados en cliente | 18 |
| 6 | Modificación de respuesta | 19 |
| 7 | Mensajes de control (errores más comunes)..... | 20 |
| 7.1 | Unauthorized access: INVALID_PAGE_ID | 20 |
| 7.2 | Unauthorized access: INVALID_PARAMETER_NAME | 20 |
| 7.3 | Unauthorized access: INVALID_PARAMETER_VALUE | 20 |
| 7.4 | Unauthorized access: INVALID_HDIV_PARAMETER_VALUE | 21 |
| 7.5 | Unauthorized access: INVALID_HID_VALUE | 21 |
| 7.6 | Unauthorized access: INVALID_BODY_FORMAT | 21 |
| 7.7 | Unauthorized access: URL not allowed | 21 |
| 8 | Cambios importantes..... | 22 |

1 Introducción

Hdiv es una solución de seguridad para aplicaciones web y APIs que cuenta con dos productos: uno para detectar vulnerabilidades de seguridad en el 'software' y otro que protege ese 'software' para que los sistemas estén securizados y poder automatizar esa seguridad sin que dependa de la acción directa de las personas.

UDA, como conjunto de utilidades para el desarrollo de aplicaciones de Gobierno Vasco se ha integrado la solución de protección que propone Hdiv para mejorar estructuralmente el desarrollo seguro de las aplicaciones. web y ofrece una librería de seguridad que integra x38 y Hdiv.

La integración de UDA y Hdiv a través de una librería parametriza el comportamiento de Hdiv y posibilita el uso de mecanismos durante el desarrollo de software. En los siguientes apartados se describen los pasos necesarios para la instalación y la descripción de cómo adoptar esos mecanismos.

Además de este documento se puede encontrar más información en la [documentación oficial de Hdiv](#).

2 Validaciones

La librería de Hdiv realiza tres tipos de validaciones:

- Control de acceso
- Validaciones de editables
- Validaciones de integridad

La configuración de seguridad se podrá modificar desde la consola Hdiv teniendo ésta prioridad sobre la definida en la propia aplicación.

2.1 Control de acceso

Consiste en determinar qué URLs quedan accesibles para el usuario.

Por defecto ninguna URL será permitida a no ser que se habilite desde la parte servidora de la aplicación.

Más adelante se muestra cómo habilitar los accesos.

2.2 Validación de editables


Se trata de validar los datos enviados en campos de texto.

Existe una validación por defecto que podrá ser completada por el desarrollador.

2.3 Validación de integridad

La librería comprueba que los valores enviados en campos con opciones finitas como radios, checkboxes, combos, etcétera son parte de las opciones proporcionadas desde el servidor.

3 Instalación de la librería

La librería de Hdiv se ofrece integrada con la librería de utilidades de  (x38ShLibClasses) a partir de la versión 5. Al hacer uso de la x38ShLibClasses en el desarrollo de una aplicación se cargarán las dependencias necesarias para incluir la librería Hdiv.

3.1 Importación de dependencias

La carga de la librería se realiza mediante la configuración en el fichero de dependencias [codapp]EAR/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.ejje.x21a</groupId>
    <artifactId>x21aEAR</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>x21aEAR</name>
    <url>http://maven.apache.org</url>
    <properties>
        <org.springframework.version>4.3.22.RELEASE</org.springframework.version>
        <org.springframework.security.version>4.2.11.RELEASE</org.springframework.security.version>
        <org.logback.version>1.2.3</org.logback.version>
        <org.slf4j.version>1.7.30</org.slf4j.version>
        <org.jackson.version>2.7.9.5</org.jackson.version>
        <org.apache.tiles.version>3.0.8</org.apache.tiles.version>
        <com.ejje.x38.version>5.0.2-RELEASE</com.ejje.x38.version>
        <hdivsecurity.version>2.4.1</hdivsecurity.version>
    </properties>
    <dependencies>
        <!-- Hdiv -->
        <dependency>
            <groupId>com.hdivsecurity</groupId>
            <artifactId>hdiv-for-services</artifactId>
            <version>${hdivsecurity.version}</version>
            <exclusions>
                ...
            </exclusions>
        </dependency>
```

3.2 Configurar filtro y listener

Para habilitar el uso de la librería Hdiv en los proyectos de desarrollo,

- se configurará un filtro y un listener en el fichero \codAppNombreAppWar\WebContent\WEB-INF\web.xml

Importante: El filtro de Hdiv debe ser el primero en registrarse. Para ello se situará por delante del filtro de UDA.

```
<!-- Hdiv Init Listener -->
<listener>
  <listener-class>org.hdiv.listener.InitListener</listener-class>
</listener>

<!-- Hdiv Validator Filter -->
<filter>
  <filter-name>ValidatorFilter</filter-name>
  <filter-class>org.hdiv.filter.ValidatorFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>ValidatorFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

- se cambiará el archivo mvc-config.xml para que el base scan encuentre el archivo de configuración.

3.3 Configurar clase de parametrización

Activado el filtro y el listener de Hdiv, será necesario añadir una clase de configuración para parametrizar el comportamiento de la librería dentro de la aplicación. A su vez esta clase extenderá de la clase UDA4HdivConfigurerAdapter incluida en la librería x38 donde se particulariza la adaptación de Hdiv en los desarrollos con UDA.

Un ejemplo básico sería el siguiente:

```
@Configuration
public class UDA4HdivConfig extends UDA4HdivConfigurerAdapter {

    @Override
    protected String getHomePage() {
        return "/";
    }

    @Override
```

```
protected String getLoginPage() {  
    return "/loginPage";  
}  
  
protected String getDashboardUser() {  
    return "dashboard-admin";  
}  
  
protected String getDashboardPass() {  
    return "password";  
}  
  
@Override  
public void addCustomExclusions(final ExclusionRegistry registry) {  
}  
  
@Override  
public void addCustomRules(final RuleRegistry registry) {  
}  
  
@Override  
public void customConfigureEditableValidation(final ValidationConfigurer validationConfigurer) {  
}  
}
```

Más adelante se explicará con más detalle la configuración de la librería.

El método `getHomePage` establece la URL por la que se accede a la aplicación.

En `getLoginPage` se especifica la URL de la página de login.

En `getDashboardUser` y `getDashboardPass` se especifica el usuario y contraseña de acceso al dashboard de Hdiv. En este dashboard se muestra la información de seguridad relativa a la aplicación.

Para acceder a esta vista de información:

```
{appContextPath}/hdiv-services/dashboard
```

Para añadir exclusiones de validación se hará uso del método `addCustomExclusions`.

Para crear nuevas reglas de validación, se hará uso del método `addCustomRules`, donde se especificará un nombre de validación y los patrones aceptados y/o rechazados.

Un ejemplo de validación en la que se aceptan únicamente valores numéricos sería:

```
registry.addRule("numerico").acceptedPattern("^([0-9]+)$");
```


Para asignar las reglas existentes a URLs o parámetros se hará uso de `customConfigureEditableValidation`.

Un ejemplo de asignación de la regla de valores booleanos sería el siguiente:

```
validationConfigurer  
    .addValidation("/.*")  
    .forParameters("parameter.name")  
    .rules("boolean")  
    .target(ValidationTargetType.CLIENT_PARAMETERS);
```

El atributo `CLIENT_PARAMETERS` se refiere a parámetros creados directamente en el cliente sin ser enviados desde el servidor (javascript).

4 Mecanismos de protección

A continuación, se detallan los mecanismos de protección que habilita la librería de Hdiv y que habrá que tener en cuenta en el desarrollo de la aplicación.

4.1 Habilitar acceso

Para permitir el acceso a URLs será necesario que estas estén creadas mediante el tag de spring `spring:url` o permitidas desde el controlador. El método aconsejado es mediante el tag de spring, siendo la habilitación desde el controlador el método utilizado para URLs creadas en cliente (javascript).

Ejemplo Caso 1 (Recomendado):

```
<spring:url value="/patrones/feedback" var="feedback" htmlEscape="true"/>
<a class="dropdown-item" href="${feedback}">
    <i class="mdi mdi-message-alert" aria-hidden="true"></i>
    <spring:message code="feedback" />
</a>
```

Ejemplo Caso 2:

```
@UDALink(name = "getAllIds")
@GetMapping(value = "/allIds")
public @ResponseBody
List<Resource<Usuario>> getAllIds(
    @RequestParam(value = "q", required = true) String param,
    @RequestParam(value = "c", required = true) boolean startswith) {
    TableUsuarioController.logger.info("[GET - find_ALL_ID] : Obtener CPs de Usuario");
    Usuario usuario = new Usuario(param);
    return ResourceUtils.fromListToResource(this.tableUsuarioService.findAllIds(usuario, startswith));
}
```

Las peticiones realizadas a URLs no generadas de las siguientes maneras serán bloqueadas.

4.1.1 Spring:url

Esta forma de generar las URLs hace que el servidor sea el encargado de crearlas y de registrarlas. En las URLs se añadirá un parámetro de seguridad encargado de identificar el enlace.

Incorrecta:

```
<a class="nav-link" href="/appContext/url">
```

Adecuada:

```
<spring:url value="/url" var="url"/>
    <a class="nav-link" href=${url}>
    ...
```

Las URLs que coinciden con el siguiente listado están excluidas de validación en la configuración de la librería, y por lo tanto no requieren del tag de spring.

```
/scripts/.*, /styles/.*, /fonts/.*, /error, /*.gif
```

4.1.2 Link allower

Se ha implementado una herramienta que permite habilitar URLs desde el controlador:
`@UDALink`.

Esta herramienta se utiliza en forma de anotación.
Consta de dos partes, identificación del link y accesos permitidos.

Al acceder al método del controlador con esta anotación, se habilitará el acceso a los links definidos en `linkTo`.

Por ejemplo, un método anotado de la siguiente manera permite el acceso a las peticiones de los métodos marcados como `UDALink` “edit” y “remove”.

```
@UDALink(name = "get", linkTo = { @UDALinkAllower(name = "edit" ),
                                @UDALinkAllower(name = "remove" )})
```

Una condición para el uso de esta anotación, es que en el caso de existir **path** variables en el **mapping** de los métodos a habilitar, el nombre de las **path variables** debe **coincidir** con el nombre de atributo del objeto enviado en la response.

Solo se permitirá el acceso a los métodos que coincida esta regla, es decir, siguiendo con el ejemplo anterior no se podrá editar o eliminar ningún objeto que no haya sido devuelto en el método con nombre “get”.

```
@UDALink(name = "remove")
@RequestMapping(value =("/{id}", method = RequestMethod.DELETE)
```

El atributo `id` debe existir en el objeto devuelto por el método “get”.

Si el método al que se quiere dar acceso se encuentra en otro controlador, se hará uso del atributo “`linkClass`” de la anotación `UDALinkAllower` dentro de `UDALink`.

```
@UDALink(name = "get",
          linkTo = { @UDALinkAllower(name = "edit",
                                    linkClass = CustomOtherController.class) })
```

Hay casos en los que no todas las entidades devueltas por el método del controlador deben tener acceso a los links anotados.

Para discriminar parte de estos recursos se hará uso del atributo “`allower`”.

```
@UDALink(name = "get",
          linkTo = { @UDALinkAllower(name = "edit",
                                    allower = CustomController.class) })
```

Este atributo define la clase en la que se encuentra el predicado con la lógica de clasificación.

El predicado debe ser un método público y estático que devuelva un objeto de tipo `LinkPredicate<Tipo_De_Entidad>` con la implementación del método “test”. El método test recibe el objeto “LinkInfo” como parámetro.

```
public class LinkInfo<T> {
    private final String linkId;
    private final T entity;
    private final HttpServletRequest request;
}
```

A continuación, se muestra un ejemplo de implementación:

```
public static LinkPredicate<Entidad> getEntidadLinkPredicate() {
return new LinkPredicate<Entidad>() {
    @Override
    public boolean test(LinkInfo<Entidad> linkInfo) {
        return !linkInfo.getEntity().isClosed();
    }
};
}
```

4.2 Envío de datos

El envío de datos al servidor será de dos tipos:

- Formularios con tags de Spring
- Peticiones tipo REST

4.2.1 Formularios con tags de Spring

Los formularios deben hacer uso de los tags de spring y definir el campo action.

```
<spring:url value="/url" var="url"/>
```

```
<form:form id="form_id" action="${url}">
```

Ejemplo para las tablas en la Jsp:

```
<!-- Formulario -->
<c:set value="${actionType == 'POST' ? 'add': 'edit'}" var="endpoint" />
<spring:url value="/table/${endpoint}" var="url"/>
<form:form modelAttribute="usuario" id="example_detail_form" action="${url}" method="${actionType}">
    <!-- Feedback del formulario de detalle -->
    <div id="example_detail_feedback"></div>
    <c:if test="${not empty fixedMessage}">
        <p><c:out value="${fixedMessage}"/></p>
    </c:if>
```

Es importante incluir el taglib de spring-form.

```
<%@ taglib prefix="form" uri="/WEB-INF/tld/spring-form.tld"%>
```

Los datos estáticos de los formularios como combos, radios, checkbox, etc., se deberán cargar en tiempo de renderizado. Lo más común es emplear el modelo para añadir estos datos como atributos:

```
@UDALink(name = "getApellidos")
@RequestMapping(value = "/apellidos", method = RequestMethod.GET)
public @ResponseBody List<AutocompleteComboGenericPOJO> getApellidos (

    //En el js
    editoptions: {
        source : './apellidos',
        sourceParam : {label: 'label', value: 'value'},
        menuMaxHeight: 200,
        minLength: 3,
        combobox: true,
        contains: true,
        showDefault: true
    }
}
```

Controller

```
Map<String,String> att = new LinkedHashMap<String,String>();
att.put("value1", "value1Name");
att.put("value2", "value2Name");
model.addAttribute("attname", att);
```

JSP

```
<form:select path="name" id="estado_filter_table" class="rup-combo" items="${attname}"/>
```

Controller

```
model.addAttribute("attname2", att2ValueList);
```

JSP

```
<form:select path="name2" class="rup-combo">
    <form:option value="">&nbsp;</form:option>
    <form:options items="${attname2}" itemLabel="desc" itemValue="key"/>
</form:select>
```

Es importante comprender que los datos permitidos en un formulario están asociados al “action” del mismo. Si se modifica la URL del formulario, esta no corresponderá con los parámetros permitidos y la petición será bloqueada.

La librería considera que los campos definidos como hidden no son editables, con lo que si estos sufren modificaciones (su valor no corresponde al del momento de renderizado del formulario) la petición será bloqueada.

Si hay campos ocultos en los que las modificaciones están permitidas, estos deberán ser **inputs** ocultos, no hidden.

```
<form:input path="parameter" style="display: none"/>
```

4.2.2 Envío de entidades tipo REST

Las entidades deberán ser de tipo **SecureIdContainer** o **SecureIdentifiable**. Siempre serán representadas por un identificador único.

Este identificador viajará encriptado. Se proporcionará el valor original en el campo con nombre **"nid"**.

SecureIdentifiable: En caso de que la entidad posea un atributo llamado **"id"** o requiera de clave compuesta. Se crearán los métodos **getId** y **setId**.

SecureIdContainer: Si la entidad tiene un atributo identificativo (código cuyo nombre de atributo es distinto de **"id"**) o si el objeto no es una entidad en sí misma, pero contiene una, esta implementará **SecureIdentifiable**.

Para identificar el atributo identificador, se anotará éste como **@TrustAssertion(idFor = EntidadActual.class)**.

Ejemplo de **SecureIdContainer**.

Clase entidad:

```
public class Entidad implements java.io.Serializable, SecureIdContainer {
    @TrustAssertion(idFor = Entidad.class)
    private Integer codEntidad;
}
```

Clase que no es una entidad en sí, pero hace uso de una:

```
public class EntidadDAO implements java.io.Serializable, SecureIdContainer {
    @TrustAssertion(idFor = Entidad.class)
    private Integer codEntidad;
}
```

Al enviar las entidades a cliente se harán en forma de **Resource**.

```
new Resource<Entidad>(entidad);
```

```
public @ResponseBody Resource<Entidad> get(@PathVariable String id)
```

```
public @ResponseBody TableResourceResponseDto<Entidad> filter()
```

La librería **x38** hace uso de la clase **TableResourceResponseDto** para enviar las entidades como **Resources**. Esta clase es compatible con el componente **tabla**.

4.3 Path variables

Las URLs de peticiones que hagan uso de **path variables** deberán cumplir con el siguiente patrón:

La última variable corresponderá al identificador de la entidad.

Las variables en la URL que correspondan a la entidad y no estén modificados deberán tener el mismo nombre que el atributo de la propiedad.

Se recomienda que las variables estáticas de la URL sean una enumeración para controlar el valor de los mismos.

Por ejemplo, una variable con nombre “estado” cuyos valores se esperan entre “activo” e “inactivo”, se recomienda que sea un enumerado al menos en el path.

En caso de que el valor de una variable no quiera ser validado por integridad, se marcará en el controlador de la siguiente manera:

```
@PathVariable @TrustAssertion(idFor = NoEntity.class) Integer value
```

5 Configuración

La librería ofrece la posibilidad de añadir configuración a la aportada por defecto. Las validaciones se configurarán desde la clase de configuración que extiende de UDA4HdivConfigurerAdapter.

A continuación, se exponen las opciones de configuración.

5.1 Url exclusions - Start Pages

Las URLs añadidas como url exclusions no serán validadas.

Se establecerá la URL (puede contener expresiones regulares) y se podrá especificar el verbo del método http al que aplica.

@Override

```
public void addCustomExclusions(final ExclusionRegistry registry) {  
    registry.addUrlExclusions("/url.html").method("GET");  
}
```

Como parte especial de URLs excluidas de validación se encuentran las URLs de entrada a la aplicación y login. Éstas se configuran de la siguiente manera:

@Override

```
protected String getHomePage() {  
    return "/";  
}
```

@Override

```
protected String getLoginPage() {  
    return "/loginPage";  
}
```

5.2 Text fields validation

Se permite especificar la validación de valor de parámetro por nombre de parámetro y URL.

Esta regla se compone de la URL en la que se aplica la validación, el patrón aceptado, el rechazado y el nombre que se da a la regla.

En la regla se podrá especificar el patrón aceptado y/o el rechazado. Ejemplo configuración:

```
@Override
public void configure(SecurityConfigBuilder builder) {

    builder.textFieldIntegration(TextFieldIntegration.LIBRARY_ONLY);
}
```

5.2.1 AcceptedPattern

Patrón de valores aceptados en el parámetro.

Se compone de expresiones regulares.

```
@Override
public final void addRules(final RuleRegistry registry) {
    registry.addRule("numeric").acceptedPattern("^([0-9]+)$");
    registry.addRule("text").acceptedPattern("[a-zA-Z0-9@.\\-~]*").rejectedPattern("(\\s|\\s)*(--)(\\s|\\s)*");
    registry.addRule("fulltext").acceptedPattern("[a-zA-Z0-9@.\\-~]*").rejectedPattern("(\\s|\\s)*(--)(\\s|\\s)*");
    registry.addRule("valueList").acceptedPattern("\\[[^()]*\\]");
    registry.addRule("url").acceptedPattern("^((https?|ftp|file)://[-a-zA-Z0-9+&@#/%?~_!:,.;]*[-a-zA-Z0-9+&@#/%~_!])");
    registry.addRule("partialUrl").acceptedPattern("[a-zA-Z0-9@.\\-~]*").rejectedPattern("(\\s|\\s)*(--)(\\s|\\s)*");
    registry.addRule("boolean").acceptedPattern("(\\w|^)(true|false)(\\w|$)");
    registry.addRule("order").acceptedPattern("(\\w|^)(asc|desc)(\\w|$)");
    registry.addRule("locale").acceptedPattern("(\\w|^)(es|eu|en|fr)(\\w|$)");

    addCustomRules(registry);
}
```

5.2.2 RejectedPattern

Patrón de valores rechazados en el parámetro.

Se compone de expresiones regulares.

A continuación, se muestra un ejemplo de regla de validación de editables.

```
@Override
public void addCustomRules(final RuleRegistry registry)
    registry.addRule("customValidation")
        .acceptedPattern("[a-zA-Z0-9@.\\-~]*");
}

@Override
public void customConfigureEditableValidation(final ValidationConfigurer validationConfigurer){
    validationConfigurer.addValidation(".").rules("customValidation");
}
```

5.3 Parameter exclusions - Start Parameters (NO recomendado)

Al igual que se ofrece una opción para evitar la validación por URLs, se permite excluir las validaciones de parámetros en concreto.

Estas exclusiones pueden especificarse por URL siguiendo una expresión regular.

A continuación se muestra un ejemplo de configuración para la exclusión del parámetro “paramName” para todas las URLs que empiezan por “section/”.

```
@Override
public void addExclusions(ExclusionRegistry registry) {
    registry.addParamExclusions("paramName").forUrls("/section/.*");
}
```

5.4 Parámetros generados en cliente

En ocasiones hay parámetros que son generados en el cliente y por lo tanto desconocidos para la parte servidora.

Una petición con este tipo de parámetros sería bloqueada por la librería por tratarse de datos no autorizados por el servidor.

Si es necesario que estos parámetros sigan siendo creados en la parte cliente (js, html) se deberá añadir una validación indicando que son parámetros creados en cliente esperados por el servidor.

Esta configuración se hará de la siguiente manera:

Parámetro generado en cliente con validación por defecto:

```
@Override
public void customConfigureEditableValidation(final ValidationConfigurer validationConfigurer){
    validationConfigurer.addValidation("/.*")
        .forParameters("paramName")
        .target(ValidationTargetType.CLIENT_PARAMETERS;
}
```

Parámetro generado en cliente con validación personalizada:

```
@Override
public void customConfigureEditableValidation(final ValidationConfigurer validationConfigurer){
    validationConfigurer.addValidation("/.*")
        .forParameters("paramName")
        .rules("customValidation")
        .target(ValidationTargetType.CLIENT_PARAMETERS).disableDefaults();
}
```

6 Modificación de respuesta

El objeto response que se usa en las tablas ha cambiado y No se devuelve un Objeto sino un Resource del objeto en este caso Usuario.

```
@UDALink(name = "edit", linkTo = { @UDALinkAllower(name = "filter") })
@RequestMapping(value = "/edit", method = RequestMethod.PUT)
public @ResponseBody Resource<Usuario> edit(@RequestBody Usuario usuario) {
    Usuario usuarioAux = this.tableUsuarioService.update(usuario);
    logger.info("Entity correctly updated!");
    return new Resource<Usuario>(usuarioAux);
}
```



7 Mensajes de control (errores más comunes)

Cuando Hdiv bloquea una petición, lo justifica devolviendo en la respuesta un mensaje con el motivo. A continuación, se informa acerca de los errores más frecuentes:

7.1 Unauthorized access: INVALID_PAGE_ID

```
X Headers Payload Preview Response Initiator Timing Cookies
▼ {message: "HDIV. Unauthorized access.",...}
  ▼ errors: [{type: "INVALID_PAGE_ID", rule: "AUTOMATED_REAL_TIME_WHITELISTING", parameterName: null,...}]
    ▼ 0: {type: "INVALID_PAGE_ID", rule: "AUTOMATED_REAL_TIME_WHITELISTING", parameterName: null,...}
      parameterName: null
      parameterValue: null
      rule: "AUTOMATED_REAL_TIME_WHITELISTING"
      type: "INVALID_PAGE_ID"
      message: "HDIV. Unauthorized access."
```

Este error sucede porque se define una URL sin el tag de Spring URL.

7.2 Unauthorized access: INVALID_PARAMETER_NAME

Cuando el formulario contiene un campo que no existe en la entidad o un campo que no haya sido renderizado en la JSP:

```
X Headers Payload Preview Response Initiator Timing Cookies
▼ {message: "HDIV. Unauthorized access.", errors: [...]}
  ▼ errors: [...]
    ▼ 0: {type: "INVALID_PARAMETER_NAME", rule: "AUTOMATED_REAL_TIME_WHITELISTING", parameterName: "nombre2",...}
      parameterName: "nombre2"
      parameterValue: null
      rule: "AUTOMATED_REAL_TIME_WHITELISTING"
      type: "INVALID_PARAMETER_NAME"
      message: "HDIV. Unauthorized access."
```

7.3 Unauthorized access: INVALID_PARAMETER_VALUE

Cuando se intenta añadir desde el cliente un campo nuevo en un formulario:

```
X Headers Payload Preview Response Initiator Timing Cookies
▼ {message: "HDIV. Unauthorized access.",...}
  ▼ errors: [{type: "INVALID_PARAMETER_VALUE", rule: "AUTOMATED_REAL_TIME_WHITELISTING", parameterName: "id",...}]
    ▼ 0: {type: "INVALID_PARAMETER_VALUE", rule: "AUTOMATED_REAL_TIME_WHITELISTING", parameterName: "id",...}
      parameterName: "id"
      parameterValue: "aa"
      rule: "AUTOMATED_REAL_TIME_WHITELISTING"
      type: "INVALID_PARAMETER_VALUE"
      message: "HDIV. Unauthorized access."
```

Si no está definido en la entidad como atributo o si estando definido en la entidad, no aparece declarado de forma explícita en el formulario, Hdiv bloquearía las peticiones.

7.4 Unauthorized access: INVALID_HDIV_PARAMETER_VALUE

Si el argumento HDIV_STATE ha sido modificado o se ha perdido:

```

× Headers Payload Preview Response Initiator Timing Cookies
▼ {message: "HDIV. Unauthorized access.",...}
  ▼ errors: [{type: "INVALID_HDIV_PARAMETER_VALUE", rule: "AUTOMATED_REAL_TIME_WHITELISTING",...}]
    ▼ 0: {type: "INVALID_HDIV_PARAMETER_VALUE", rule: "AUTOMATED_REAL_TIME_WHITELISTING",...}
      parameterName: "_HDIV_STATE_"
      parameterValue: "54589827-19-10-2979C0D002B612AC91F928907E1"
      rule: "AUTOMATED_REAL_TIME_WHITELISTING"
      type: "INVALID_HDIV_PARAMETER_VALUE"
      message: "HDIV. Unauthorized access."

```

7.5 Unauthorized access: INVALID_HID_VALUE

Si el identificador de una entidad es editado, este es el error que devolverá Hdiv:

```

message: "HDIV. Unauthorized access."
▼ errors: [{...}]
  ▼ 0: Object { type: "INVALID_HID_VALUE", rule: "AUTOMATED_REAL_TIME_WHITELISTING", target: "/x21aAppWar/table/edit", ... }
    type: "INVALID_HID_VALUE"
    rule: "AUTOMATED_REAL_TIME_WHITELISTING"
    target: "/x21aAppWar/table/edit"
    parameterName: "hid-EntityId [id=553, class=Modelos]"
    parameterValue: "1-553-627F83BE-68-0-DED44B4F0D10A6A45A89E2F364854933"
    originalParameterValue: null

```

7.6 Unauthorized access: INVALID_BODY_FORMAT

Se produce cuando se detecta una malformación en el body (JSON) y como consecuencia, ocurre una excepción en la validación:

```

message: "HDIV. Unauthorized access."
▼ errors: [{...}]
  ▼ 0: Object { type: "INVALID_BODY_FORMAT", rule: "AUTOMATED_REAL_TIME_WHITELISTING", target: "/x21aAppWar/table/edit", ... }
    type: "INVALID_BODY_FORMAT"
    rule: "AUTOMATED_REAL_TIME_WHITELISTING"
    target: "/x21aAppWar/table/edit"
    parameterName: null
    parameterValue: null
    originalParameterValue: null

```

7.7 Unauthorized access: URL not allowed

Si la URL no ha sido declarada como permitida desde el controlador:

```

× Headers Payload Preview Response Initiator Timing Cookies
▼ {message: "HDIV. Unauthorized access.",...}
  ▼ errors: [{type: "Error validating request POST /table/editForm. URL not allowed",...}]
    ▼ 0: {type: "Error validating request POST /table/editForm. URL not allowed",...}
      parameterName: null
      parameterValue: null
      rule: "AUTOMATED_REAL_TIME_WHITELISTING"
      type: "Error validating request POST /table/editForm. URL not allowed"
      message: "HDIV. Unauthorized access."

```

8 Cambios importantes

Hasta ahora, había sido una práctica común generar campos en el cliente que posteriormente se enviarían al servidor o incluso generar y editar identificadores de los registros. Con la inclusión de la librería de Hdiv, todas estas prácticas quedan prohibidas y serán bloqueadas en caso de ser llevadas a cabo.

Un cambio necesario para todas las aplicaciones que hasta ahora habían permitido la creación o modificación de identificadores desde el cliente, es la creación de secuencias en base de datos para que la inserción de registros puede ser llevada a cabo correctamente desde el DAO.

Un ejemplo de una secuencia utilizada en la X21A:

```
CREATE SEQUENCE "X21B"."USUARIO_SEQ" MINVALUE 1 MAXVALUE 99999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE NOPARTITION;
```

Un ejemplo de la inserción de identificadores a partir de una secuencia usada en la X21A:

```
public Usuario add(Usuario usuario) {
    // Obtenemos el identificador de la entidad mediante una secuencia
    final String nextId = jdbcTemplate.queryForObject("SELECT USUARIO_SEQ.NEXTVAL FROM DUAL", String.class);
    usuario.setId(nextId);

    String query = "INSERT INTO USUARIO (ID, NOMBRE, APELLIDO1, APELLIDO2, EJIE, FECHA_ALTA, FECHA_BAJA, ROL, FECHA_MODIF) VALUES (?, ?, ?, ?, ?, ?, ?, ?, sysdate)";
    this.jdbcTemplate.update(query, usuario.getId(), usuario.getNombre(), usuario.getApellido1(), usuario.getApellido2(), usuario.getEjia(), usuario.getFechaAlta(), usuario.getFechaBaja(), usuario.getRol());
    return usuario;
}
```