Rapido Documentation

Versión 1.0

Makina Corpus

25 de octubre de 2018

Índice general

Feliz hacking con Plone

CAPÍTULO 1

¿Para qué?

Creación de un pequeño formulario capaz de enviar un correo electrónico, o para almacenar algunos datos, generando alguna información adicional sobre una página y la inserción donde quiera: con Plone este tipo de tareas son complejas para los expertos, y casi imposible para los principiantes.

rapido.plone permite a cualquier desarrollador tener un poco de conocimiento de HTML y un poco de conocimiento de Python para implementar elementos personalizados e insertarlos donde quieran en su sitio Plone.

CAPÍTULO 2

¿Cómo?

La interfaz única para crear aplicaciones con rapido.plone es la herramienta **Plone theming**.

Implica que se puede lograr en el **sistema de archivos** o mediante el **editor de temas en línea**.

Una aplicación Rapido es sólo una parte de nuestro tema actual; Puede ser importado, exportado, copiado, modificado, etc. como el resto del tema.

Por otra parte, podemos utilizar Diazo extensivamente para incorporar nuestra aplicación en el diseño de Plone fácilmente.

Contenidos:

2.1 Plone la forma fácil

Crear un sitio con Plone es fácil, pero desarrollar con Plone puede parecer desalentador.

De hecho, puede ser fácil, y no necesariamente implica aprender mucho sobre varios frameworks complejos.

Aquí están los **conceptos básicos de desarrollo fácil con Plone**.

2.1.1 Instalarlo

La instalación de Plone es muy sencilla (consulte la documentación de instalación).

2.1.2 Aplicarle Temas

Diazo es el motor de tematización de Plone. Diazo tiene un enfoque brillante para realizar temas: **se aplica en la parte superior de Plone, no dentro**.

De hecho, Plone produce páginas de contenido y Diazo puede aplicar cualquier tema sobre la marcha a esas páginas. Así que no necesitamos saber nada acerca de los mecanismos internos de Plone para discutirlo.

Diazo sólo requiere un **tema estático regular** (archivos HTML, CSS, JS, etc.) y **algunas reglas de mapeo** (especificadas en un archivo llamado `rules.xml`) que permite especificar donde cada parte de nuestro Las páginas de contenido de Plone deben encajar en nuestro diseño estático.

El tema de Diazo se puede construir directamente desde la interfaz de Plone en el editor de Temas. El tema predeterminado de Plone 5 (llamado Barceloneta) se puede copiar y podemos modificar lo que queramos en esta copia.

La copia también se puede exportar como archivo `.zip` e importar de nuevo al mismo sitio (para restaurar una versión anterior) o en otro sitio (por ejemplo, para implementar un nuevo tema desde el sitio web de desarrollo hasta el sitio web de producción).

Si no nos sentimos cómodos con la gestión de nuestra implementación de temas en una interfaz basada en web, también podemos almacenarlo en nuestro servidor en la carpeta de instalación de Plone:

```
$INSTALL_FOLDER/resources/theme/my-theme
```

2.1.3 Extiéndalo

Plone se puede extender de dos maneras.

Podemos instalar complementos desarrollados por la comunidad Plone.

Y también podemos crear nuestros propios tipos de contenido específico usando Dexterity.

Dexterity es el framework de tipo de contenido para Plone y permite crear nuevos tipos de contenido a través de la interfaz web de Plone.

Al igual que con Diazo, podemos exportar lo que se ha creado en línea, para poder importarlo de nuevo más tarde o importarlo en otro servidor.

2.1.4 Personalizarlo

Una vez que hayamos cambiado el diseño con Diazo, es posible que deseemos volver a organizar o enriquecer el diseño de contenido ellos mismos.

Mosaic es la **solución perfecta para manipular el diseño del contenido**: podemos mover los elementos existentes (como el título, la descripción, etc.), pero también agregar nuevos.

Una vez que se crea un diseño, se puede exportar y copiar en nuestro archivo Diazo `manifest.cfg` para que esté disponible como un nuevo diseño para nuestros usuarios.

Diazo y Mosaic nos permite controlar completamente cómo se muestra la información en nuestro sitio web, pero no permiten cambiar el comportamiento de Plone, como añadir nuevas características, nueva información calculada dinámicamente, etc.

Se puede lograr con **Rapido** (como se explica en *Tutorial*), con un conocimiento muy básico de HTML y Python (así, todavía, no hay necesidad de aprender sobre los diferentes frameworks de Plone).

Nuestros desarrollos de Rapido se gestionan en nuestra carpeta de temas existente, así que aquí nuevamente podemos trabajar en línea en el editor de temas de Plone, o en la carpeta `/resources/theme`.

Rapido proporciona un fácil acceso a la API de Plone. La API de Plone reúne en un único módulo muchas herramientas diferentes de Plone que permiten buscar contenidos, crear contenidos, acceder a la información de perfiles de usuarios, etc. Hace que las características internas de Plone sean mucho más accesibles y desarrollar con Rapido podría ser una buena oportunidad para descubrir Plone a través de Su API.

2.1.5 Y si queremos...

Podría ser suficiente para cubrir casi todo lo que necesitemos implementar en nuestro sitio de Plone.

Pero si en algún momento nos sentimos lo suficientemente cómodos con el entorno técnico de Plone, y si queremos aprender más, entonces podríamos considerar la posibilidad de crear nuestro propio complemento de Plone.

Nuestro complemento manejará nuestro tema Diazo (incluyendo nuestros desarrollos Rapido), nuestras definiciones de tipo de contenido Dexterity y todas nuestras configuraciones.

Está debidamente documentado en la documentación de Plone, y en el entrenamiento de Plone también podría ser muy útil.

2.2 Instalación

Instalar Plone, entonces modifique el archivo `buildout.cfg` para agregar Rapido como una dependencia:

```
eggs =
    ...
    rapido.plone

[versions]
plone.resource = 1.2
```

Entonces, ejecute su buildout:

```
$ bin/buildout -N
```

2.3 Principios

2.3.1 Creación de una aplicación Rapido

Estos son los pasos básicos para crear una aplicación Rapido:

- vaya a la carpeta de temas (en Configuración del sitio / Tema si preferimos trabajar en línea o, si prefiere trabajar en el sistema de archivos, eso puede estar en la carpeta `static` paquete de tema o en la carpeta de `resources` de su instalación de Plone, si no tiene un paquete personalizado),
- añadir una nueva carpeta llamada `rapido`,
- en la carpeta `rapido`, agregue una nueva carpeta llamada `myapp`.

Eso es todo.

Ahora, podemos implementar nuestra aplicación en esta carpeta. Aquí está una estructura típica para una aplicación rapido:

```
/rapido
    /myapp
        settings.yaml
        /blocks
            stats.html
            stats.py
            stats.yaml
            tags.html
```

(continues on next page)

(proviene de la página anterior)

```
            tags.py
            tags.yaml
```

Nota: el archivo `settings.yaml` no es obligatorio, pero permite definir los derechos de acceso si es necesario.

Nota: Una aplicación Rapido también se puede ubicar en un tema no activo (ver *Aplicación*)

Los componentes de la aplicación son `blocks`. Un bloque se define por un conjunto de 3 archivos (HTML, Python y YAML) que se encuentran en la carpeta de `blocks`.

El **archivo YAML** define los elementos. Un elemento es cualquier elemento generado dinámicamente en un bloque: puede ser un campo de formulario (input, select, etc.), pero también un botón (`ACTION`), o incluso un fragmento de HTML generado (`BASIC`).

El **archivo HTML** contiene el diseño del bloque. El mecanismo de plantilla es super simple, los elementos se adjuntan entre paréntesis, como esto: `{my_element}`.

El **archivo Python** contiene la lógica de la aplicación. Es un conjunto de funciones, cada una nombrada para el elemento o el evento al que corresponde.

Para un elemento `BASIC`, por ejemplo, necesitamos proporcionar una función con el mismo nombre que el elemento; Su valor de retorno reemplaza al elemento en el bloque.

Para un elemento `ACTION`, se supone que debemos proporcionar una función con el mismo nombre que el elemento; En este caso, se ejecutará cuando un usuario haga clic en el botón de acción.

Aquí está un ejemplo básico:

- `rapido/myapp/blocks/simpleblock.yaml`:

```
elements:
    result: BASIC
    do_something:
        type: ACTION
        label: Do something
```

- `rapido/myapp/blocks/simpleblock.html`:

```
<p>the answer to life, the universe, and everything is {result}</p>
{do_something}
```

- `rapido/myapp/blocks/simpleblock.py`:

```
def result(context):
    return "<strong>42</strong>"

def do_something(context):
    context.app.log('Hello')
```

Podemos ver nuestro bloque, visitando la siguiente URL:

http://localhost:8080/Plone/@@rapido/myapp/blocks/simpleblock

Funciona bien, pero ¿dónde está nuestro sitio de Plone ahora?

2.3.2 Insertando nuestro bloque en una página Plone

Para poner nuestro bloque en algún lugar del sitio de Plone, usamos una regla de Diazo:

```
<before css:content="#content-core">
    <include css:content="form" href="/@@rapido/myapp/blocks/simpleblock" />
</before>
```

Ahora, si visitamos cualquier página de nuestro sitio, veremos nuestro bloque.

Nota: Si queremos mostrarlo solo en la carpeta _News_, usaremos `css:if-content`:

```
<before css:content="#content-core" css:if-content=".section-news">
    <include css:content="form" href="/@@rapido/myapp/blocks/simpleblock" />
</before>
```

Consulte la documentación de Diazo para obtener más detalles.

Pero desafortunadamente, cuando hacemos clic en nuestro botón «Hacer algo», estamos redirigidos al bloque original.

To remain in the Plone page, we need to activate the `ajax` target in `rapido/myapp/blocks/simpleblock.yaml`:

```
target: ajax
elements:
    result: BASIC
    do_something:
        type: ACTION
        label: Do something
```

Ahora, cuando hacemos clic en nuestro botón, el bloque rapido se vuelve a cargar dentro de la página de Plone.

En lugar de agregar un bloque a una vista Plone existente, es posible que tengamos que proporcionar una nueva representación, asignada a una URL específica. Podemos hacerlo declarando nuestro bloque como una vista de Plone en su archivo YAML:

```
view:
  id: my-custom-view
  with_theme: true
```

Y luego llamamos a `@@my-custom-view` en cualquier contenido, como:

> http://localhost:8080/Plone/news/@@my-custom-view

Podemos crear tantas vistas como nosotros quizas necesitemos (como `@@subscribe`, `@@unsubscribe`, `@@stats`, ...).

Nota: Añadir un montón de reglas rapido en nuestro archivo `rules.xml` no es ideal.

Nosotros podríamos preferir crear un archivo `rules.xml` en nuestra carpeta `rapido/myapp` e incluirlo en nuestro archivo `rules.xml` como este:

```
<xi:include href="rapido/myapp/rules.xml" />
```

2.3.3 Ejecución de código Python

Cada función en nuestros archivos de Python toma un parámetro llamado‘‘context‘‘. El contexto da acceso a objetos útiles:

- `context.app`: la actual aplicación rapido,
- `context.block`: (si se ejecuta en un contexto de bloque) el bloque actual,
- `context.record`: (si se ejecuta en un contexto de registro) el registro actual,
- `context.request`: la petición actual a rapido (la sub-petición, si se llama de Diazo),
- `context.parent_request`: el request de la página actual (cuando se llama desde Diazo),
- `context.portal`: el objeto de portal Plone,
- `context.content`: el actual objeto de contenido Plone,
- `context.api`: la API de Plone.

Advertencia: `context` no es el `context` habitual que conocemos en Plone (como `context` en una plantilla ZPT o un PythonScript, o `self.context` en un BrowserView).

El `context` Plone suele ser el contenido actual. En Rapido podemos obtenerlo usando `context.content`.

Esto nos permite interactuar con Plone de muchas maneras, por ejemplo, podemos ejecutar consultas de catálogo, crear contenidos, cambiar el estado del flujo de trabajo, etc.

Sin embargo, se comportará como se esperaba:

- el código siempre se ejecutará con el actual derecho de acceso del usuario, por lo que se aplicarán las restricciones de acceso correspondientes de Plone,
- la política CSRF también se aplicará (por ejemplo, una operación Plone marcada como `PostOnly` fallaría si se realiza en una solicitud GET).

Nota: El código que ponemos en nuestros archivos de Python se compila y ejecuta en un entorno de espacio aislado (proporcionado por zope.untrustedpython.interpreter).

Para ayudarnos a depurar nuestro código, podemos agregar:

```
debug: true
```

en nuestro archivo `settings.yaml` de aplicación. Entonces podemos agregar algún mensaje de registro en nuestro código:

```
context.app.log("OK")
context.app.log({"something": 1)
```

y se mostrarán tanto en el registro del servidor como en la consola javascript del navegador.

2.3.4 Almacenando y recuperando datos

Una aplicación rapido proporciona un servicio de almacenamiento integrado, basado en Souper.

Nota: Souper está diseñado para almacenar (e indexar) enormes cantidades de datos pequeños (puede almacenar fácilmente los resultados de la encuesta, comentarios, clasificaciones, etc., pero no será apropiado para los archivos adjuntos, por ejemplo).

El servicio de almacenamiento Rapido almacena **registros** y los registros contienen **elementos**.

Hay 3 maneras de crear registros en Rapido:

- podemos crear registros enviando un bloque: si un bloque contiene algunos elementos de campos (como elementos `TEXT` o `NUMBER` por ejemplo), y si el bloque contiene un botón de guardar (agregando `{_save}` en su diseño), cada vez que el usuario ingrese valores en los campos y haga clic en guardar, los valores enviados se guardarán en un nuevo registro,
- podemos crear registros por código:

```
record = context.app.create_record(id='myrecord')
```

- podemos crear registros utilizando el API REST JSON de Rapido:

```
POST /:site_id/@@rapido/:app_id
Accept: application/json
{"item1": "value1"}
```

o:

```
PUT /:site_id/@@rapido/:app_id/record/:record_id
Accept: application/json
{"item1": "value1"}
```

Lo mismo ocurre con el acceso a los datos:

- podemos mostrar registros llamando a su dirección URL, y se renderizarán usando el bloque con el que fueron creados:

```
/@@rapido/myapp/record/myrecord
```

- podemos obtener un registro por código:

```
record = context.app.get_record(id='myrecord')
some_records = context.app.search('author=="JOSEPH CONRAD"')
```

- podemos obtener registros utilizando el API REST JSON de Rapido:

```
GET /:site_id/@@rapido/:app_id/record/:record_id
Accept: application/json
```

2.3.5 Integración con Plone

Además de la inserción de bloques Rapido usando Diazo en nuestro tema, también podemos integrar nuestros desarrollos Rapido en Plone utilizando:

- Mosaico: Rapido proporciona un tile para Mosaic que nos permite insertar un bloque Rapido en nuestro diseño de página.
- Reglas de contenido: Rapido proporciona una acción de regla de contenido de Plone que nos permite llamar a una función de Python de un bloque cuando ocurre un evento de Plone dado.

- Patrones de Mockup: los patrones de *modal* y *loader de contenido* pueden cargar y mostrar bloques Rapido.

Ver *Mostrando Rapido en Plone*.

2.4 Tutorial

Como construir un sistema de calificaciones de contenidos en Plone en tan solo unos minutos.

2.4.1 Objectivo

Queremos ofrecer a nuestros visitantes la posibilidad de hacer clic en un botón *«Me gusta»* en cualquier contenido de Plone, y el total de votos debe mostrarse junto al botón.

Nota: Hay un screencast que cubre los primeros pasos del tutorial de Rapido.

2.4.2 Requisitos previos

Ejecute buildout para implementar Rapido y sus dependencias (consulte *Instalación*).

Instale el complemento `rapido.plone` desde la Configuración del sitio de Plone.

2.4.3 Inicializando la aplicación Rapido

Vamos a la *Configuración del sitio* de Plone, y luego *Temas*.

Si nuestro tema activo actual no es editable en línea a través de la interfaz web de Plone (es decir, no hay botón *«Modificar tema»*), primero deberemos crear una copia editable del mismo:

- haga clic en *«Copiar»*,
- introduzca un nombre, por ejemplo *«tutorial»*.
- marque la casilla *«Inmediatamente habilitar nuevo tema»*.

De lo contrario, simplemente haga clic en el botón *«Modificar tema»*.

Podemos ver nuestra estructura de tema, que contiene archivos CSS, imágenes, HTML y reglas de Diazo.

Para inicializar nuestra aplicación Rapido llamada «rating», necesitamos:

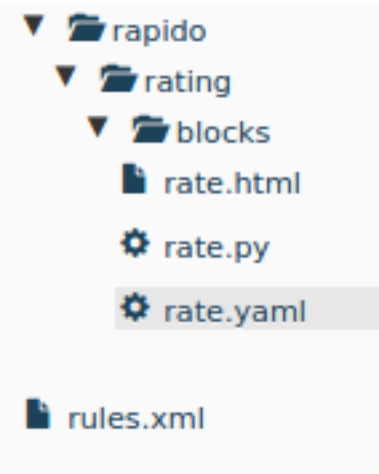
- crear una carpeta llamada `rapido` en la raíz del tema,
- en esta carpeta `rapido`, cree una carpeta llamada `rating`.

La aplicación ya está lista.

2.4.4 Creación del botón «Me gusta»

Las aplicaciones de Rapido están compuestas de **bloques**. Vamos a crear un bloque que hará que nuestro botón:

- vaya a la carpeta `rating` y cree una nueva carpeta denominada `blocks`,
- en esta carpeta de `blocks`, vamos a crear un nuevo bloque llamado `rate`. Para ello, necesitamos crear 3 archivos:



El archivo `rate.html`:

```
<i>If you like what you read, say it! {like}</i>
```

Esto nos permite implementar el diseño del bloque. Es un archivo HTML normal, pero puede contener **elementos** Rapido, entre paréntesis. En nuestro caso, tenemos un elemento, a saber `{like}`, encargado de representar el botón *«Like»*.

El archivo `rate.py`

```
def like(context):
    # nothing for now
    pass
```

Proporciona la implementación del elemento. Cada elemento del bloque tiene una función Python correspondiente que tiene el mismo id. En nuestro caso, ese es el código que se ejecutará cuando un usuario haga clic en «Like». En este momento, no hace nada, pero lo cambiaremos más tarde.

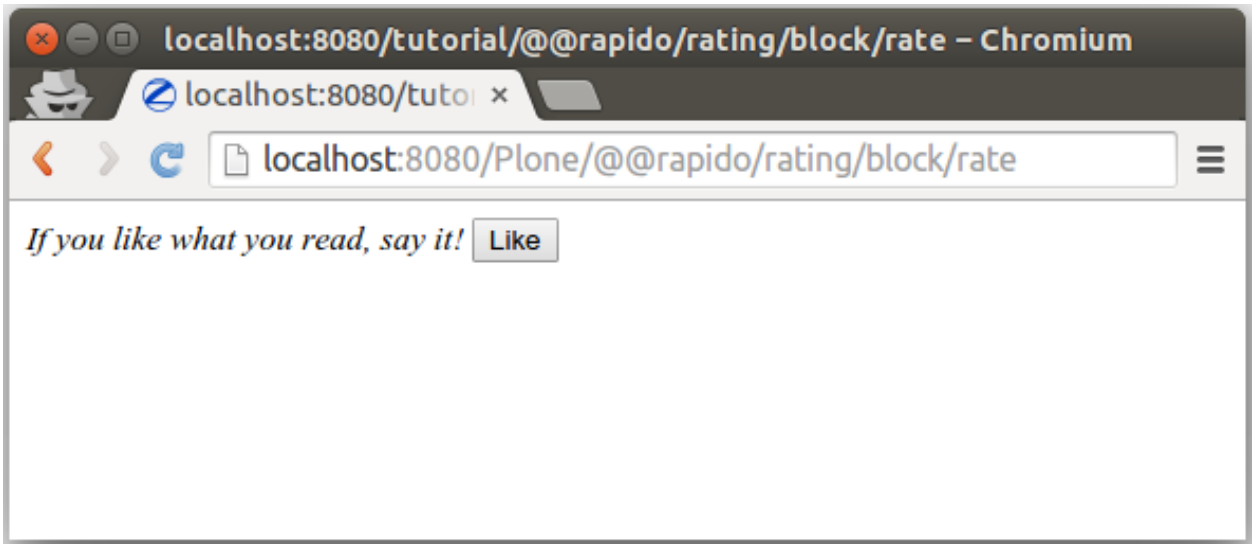
El archivo `rate.yaml`:

```
elements:
    like:
        type: ACTION
        label: Like
```

Este archivo contiene todos los ajustes necesarios para nuestro bloque. Aquí declaramos que nuestro bloque contiene un elemento denominado `like`, que es una **acción** (es decir, se renderizará como un botón), y su etiqueta mostrada es *«Like»*.

Ahora que nuestro bloque está listo, podemos verlo usando la siguiente dirección URL:

http://localhost:8080/Plone/@@rapido/rating/blocks/rate



El siguiente paso es incrustar nuestro bloque en nuestras páginas de Plone.

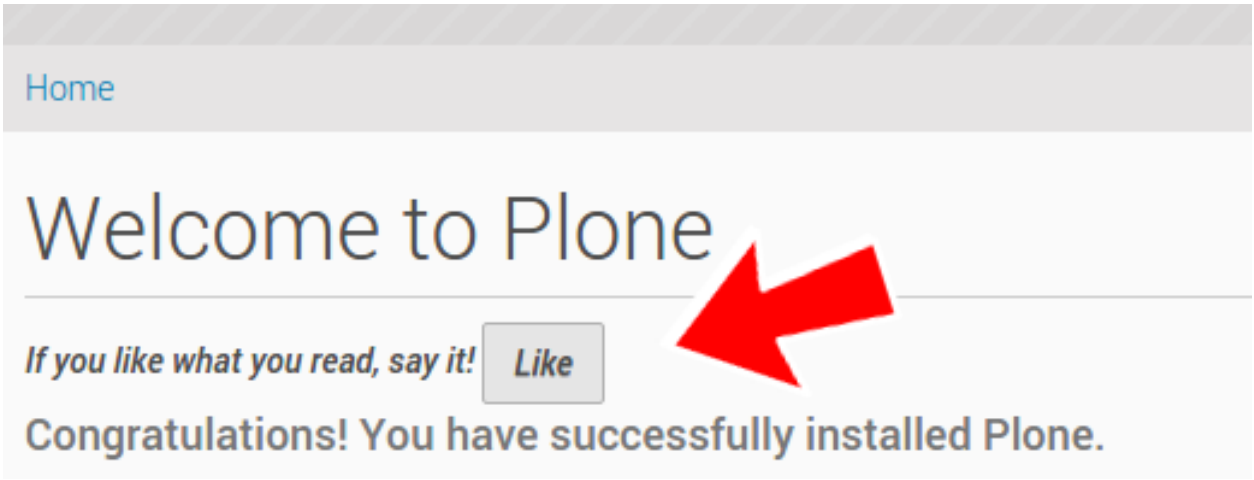
2.4.5 Insertar el bloque en páginas Plone

Para incluir nuestro bloque en algún lugar de Plone, usaremos una regla de Diazo. Abramos nuestro archivo `rules.xml` en la raíz de nuestro tema y agregue las siguientes líneas:

```
<after css:content=".documentFirstHeading">
    <include css:content="form" href="/@@rapido/rating/blocks/rate" />
</after>
```

La directiva `include` nos permite recuperar una parte del contenido; En nuestro caso, la forma HTML producida por nuestro bloque. Y la directiva `after` inserta después del título principal en nuestra página.

Por lo tanto, ahora si visitamos cualquier página de nuestro sitio de Plone, vemos nuestro bloque mostrado justo debajo del título.



Eso es bueno, pero hay un pequeño problema: cuando hacemos clic en el botón «Like», estamos redirigidos al contenido bruto del bloque, y perdemos nuestra página actual de Plone.

Vamos a arreglar eso.

2.4.6 Estando en nuestra página Plone

Si queremos permanecer en nuestra página actual después de enviar nuestro bloque, necesitamos habilitar el modo **AJAX**.

Para hacer esto, debe cambiar nuestro archivo `rate.yaml` así:

```
target: ajax
elements:
    like:
        type: ACTION
        label: Like
```

Ahora, si hacemos clic en el botón *«Like»*, el bloque se vuelve a cargar dinámicamente y nos quedamos en nuestra página actual.

2.4.7 Contando los votos

Volvamos al archivo `rate.py`, y nos enfocamos en la implementación de la función `like`.

Cuando un usuario hace clic en el botón *«Like»*, necesitamos obtener el contenido actual que el usuario votó, comprobar cuántos votos ya tiene y agregar un nuevo voto.

Rapido permite crear **registros**, por lo que crearemos un registro para cada elemento de contenido, y usaremos la ruta del contenido como un id.

Así que reemplacemos nuestra implementación actual por:

```
def like(context):
    content_path = context.content.absolute_url_path()
    record = context.app.get_record(content_path)
    if not record:
        record = context.app.create_record(id=content_path)
    total = record.get('total', 0)
    total += 1
    record['total'] = total
```

`context.content` devuelve el actual contenido de Plone y `absolute_url_path` es un método Plone que devuelve la ruta de un objeto Plone.

`context.app` permite acceder a la actual aplicación Rapido, por lo que podemos fácilmente utilizar la API de Rapido, como `create_record` o `get_record`.

Un registro Rapido contiene **elementos**. El método `get(item, default=none)` devuelve el valor del elemento solicitado o el valor predeterminado si el elemento no existe.

2.4.8 Mostrando los votos

Ahora somos capaces de almacenar votos, también queremos mostrar el *total* de votos.

Primero, vamos a cambiar el diseño del bloque en el archivo `rate.html`:

```
<p>{display}</p>
<p><i>If you like what you read, say it! {like}</i></p>
```

Así que ahora tenemos un nuevo elemento `display` en nuestro bloque.

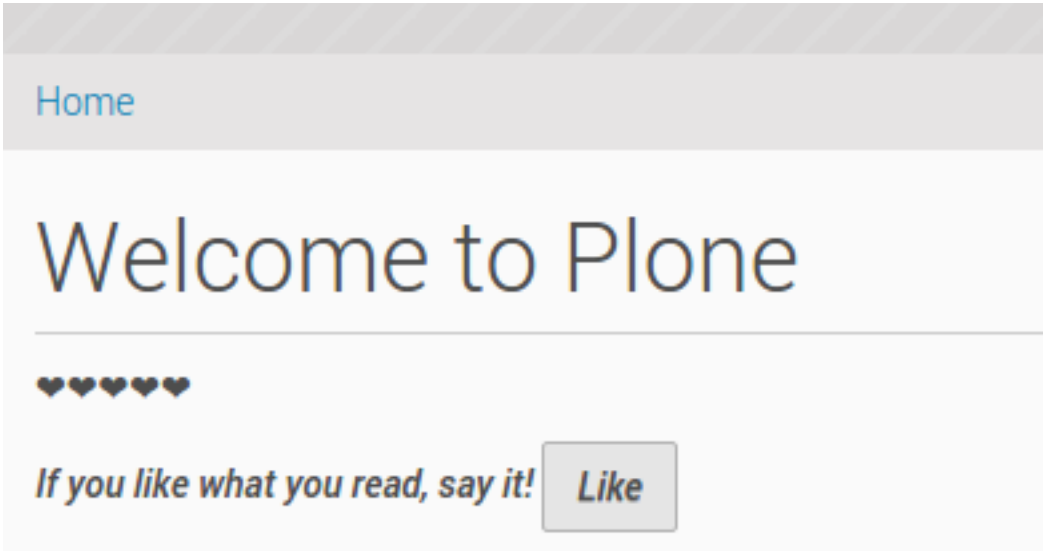
Debemos declararlo en el archivo `rate.yaml`:

```
target: ajax
elements:
    like:
        type: ACTION
        label: Like
    display: BASIC
```

Y vamos a implementarlo en el archivo `rate.py`:

```
def display(context):
    content_path = context.content.absolute_url_path()
    record = context.app.get_record(content_path)
    if not record:
        return ''
    return "&#10084;" * record.get('total', 0)
```

Obtenemos el registro correspondiente al contenido actual, y devolvemos tantos símbolos de como votos que hemos almacenado.



¡Eso es! Nuestra función de clasificación está lista para ser utilizada.

2.4.9 Depuración

Como estamos escribiendo código, nosotros podríamos (vamos a) cometer errores. En este caso, siempre es útil leer los mensajes de error devueltos por el sistema.

También es muy útil poder registrar mensajes de nuestro código, asi entendemos lo que está sucediendo exactamente cuando se ejecuta.

Rapido provee el método `context.app.log()` que registrará mensajes de cadena o cualquier objeto serializable (diccionarios, arreglos, etc.).

Los mensajes de registro y los mensajes de error están visibles en el registro del servidor (pero es posible que no podamos acceder a él), sino también en la **consola javascript** de nuestro navegador.

Lo primero que debemos hacer es activar el **modo de depuración** en nuestra aplicación. Para ello, necesitamos crear un archivo `settings.yaml` dentro de la carpeta `/rapido/rating`:

```
debug: true
```

Y ahora, cambiemos nuestra función `display`:

```
def display(context):
    content_path = context.content.absolute_url_path()
    record = context.app.get_record(content_path)
    if not record:
        return ''
    context.app.log(record.items())
    return "&#10084;" * record.get('total', 0)
```

Veremos lo siguiente en la consola de nuestro navegador:

```
Object {total: 5, id: "/tutorial"}
>
```

Imaginemos ahora que cometimos un error como olvidar el carácter `:` al final de la sentencia `if`:

```
def display(context):
    content_path = context.content.absolute_url_path()
    record = context.app.get_record(content_path)
    if not record
        return ''
    return "&#10084;" * record.get('total', 0)
```

Entonces tenemos esto en la consola:

```
Rapido compilation error - rating:
in rate, at line 14: invalid syntax
    if not record
-----------------^
>
```

2.4.10 Listado de los 5 elementos mas votados

También nos gustaría ver los 5 elementos de contenido más votados en la página principal del sitio.

Lo primero que necesitamos es indexar el elemento `total`.

Declaramos ese modo índice en el archivo `rate.yaml`:

```
target: ajax
elements:
    like:
        type: ACTION
        label: Like
    display: BASIC
    total:
        type: NUMBER
        index_type: field
```

Para indexar los valores almacenados previamente, debemos actualizar el índice de almacenamiento llamando a la siguiente dirección URL:

http://localhost:8080/Plone/@@rapido/rating/refresh

Y para asegurarnos de que los cambios futuros serán indexados, necesitamos arreglar la función `like` en el bloque `rate`: la indexación se dispara cuando llamamos al método `save` del registro:

```
def like(context):
    content_path = context.content.absolute_url_path()
    record = context.app.get_record(content_path)
    if not record:
        record = context.app.create_record(id=content_path)
    total = record.get('total', 0)
    total += 1
    record['total'] = total
    record.save(block_id='rate')
```

Ahora podemos crear un bloque para mostrar los 5 contenidos mas votados:

- el archivo `top5.html`:

```
<h3>Our current Top 5!</h3>
{top}
```

- el archivo `top5.yaml`:

```
elements:
    top: BASIC
```

- el archivo `top5.py`:

```
def top(context):
    search = context.app.search("total>0", sort_index="total", reverse=True)[:5]
    html = "<ul>"
    for record in search:
        content = context.api.content.get(path=record["id"])
        html += '<li><a href="%s">%s</a> %d &#10084;</li>' % (
            content.absolute_url(),
            content.title,
            record["total"])
    html += "</ul>"
    return html
```

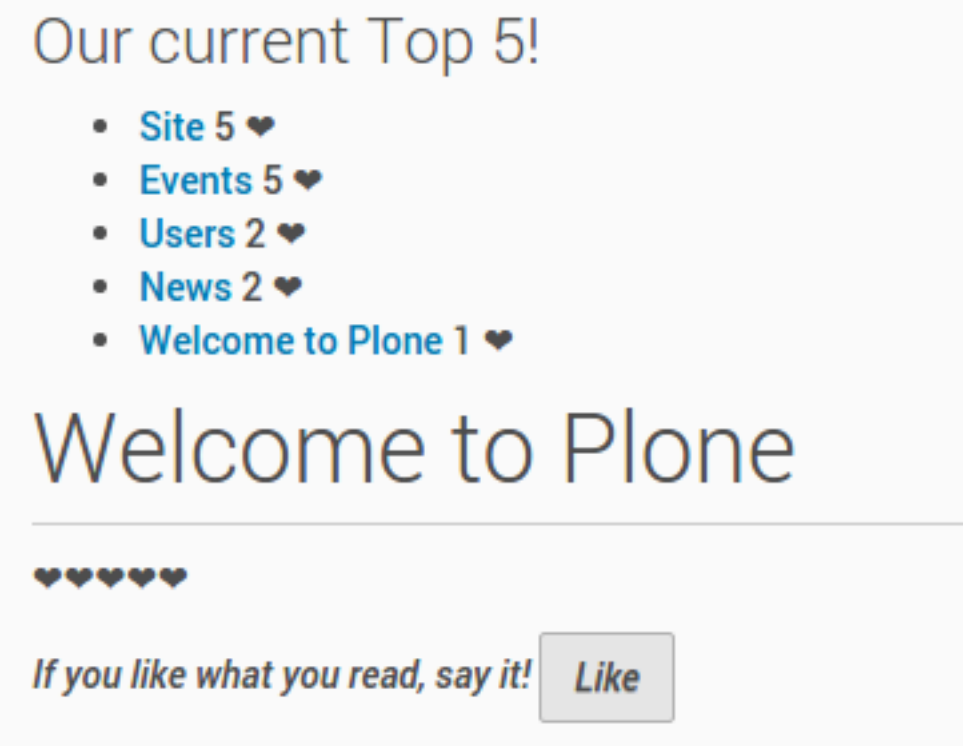
El método `search` nos permite consultar nuestros registros almacenados. Los identificadores de registro son las rutas de contenido, por lo que usando API de Plone (`context.api`), podemos obtener fácilmente el contenido correspondiente y luego obtener sus direcciones URL y títulos.

Nuestro bloque funciona ahora en la siguiente dirección URL:

http://localhost:8080/Plone/@@rapido/rating/blocks/top5

Finalmente, tenemos que insertar nuestro bloque en la página de inicio. Eso se hará en `rules.xml`:

```
<rules css:if-content=".section-front-page">
    <before css:content=".documentFirstHeading">
        <include css:content="form" href="/@@rapido/rating/blocks/top5" />
    </before>
</rules>
```



2.4.11 Creación de una nueva página para reportes

Por ahora, acabamos de añadir trozos pequeños de HTML en las páginas existentes. Pero Rapido también nos permite crear una nueva página (un desarrollador de Plone la nombraría una nueva *view* o vista).

Supongamos que queremos crear una página de reportes sobre los votos sobre el contenido de una carpeta.

Primero, necesitamos un bloque, el archivo llamado `report.html`:

```
<h2>Rating report</h2>
<div id="chart"></div>
```

Queremos que este bloque sea el contenido principal de una nueva vista.

Necesitamos declarar en un nuevo archivo YAML llamado `report.yaml`:

```
view:
    id: show-report
    with_theme: true
```

Ahora si visitamos por ejemplo:

http://localhost:8080/Plone/@@show-report

vemos nuestro bloque como contenido de la página principal.

Ahora necesitamos implementar nuestro contenido de reporte. Podríamos hacerlo con un elemento Rapido como lo hicimos en el bloque Top 5.

Vamos a cambiar nuestro enfoque e implementar una gráfico de pastel bonita utilizando la increíble librería D3js y la *API REST de Rapido*.

Necesitamos crear un archivo Javascript (`report.js`) en la carpeta `/rapido/rating`:

```
// Source: http://rapidoplone.readthedocs.io/en/latest/tutorial.html#creating-a-new-
↪page-for-reports
/* It is a feature of the RequireJS library
 * (provided with Plone by default) to load
 * our dependencies like:
 * - mockup-utils, which is a Plone internal resource,
 * - D3js (and we load it by passing its remote URL to RequireJS).
 */
require(['mockup-utils', '//d3js.org/d3.v3.min.js'], function(utils, d3) {
    /* Get the Plone getAuthenticator method
     * mockup-utils allows us to get the authenticator token
     * (with the getAuthenticator method), we need it to use
     * the Rapido REST API.
     */
    var authenticator = utils.getAuthenticator();
    // Get the local folders path
    var local_folder_path = location.pathname.split('/@@rapido')[0];
    // Get SVG element from the rapido block html named 'report.html'
    var width = 960,
        height = 500,
        radius = Math.min(width, height) / 2;

    /* d3.js Arc Generator
     * Generates path data for an arc (typically for pie charts).
     */
    var arc = d3.svg.arc()
        .outerRadius(radius - 10)
        .innerRadius(0);

    /* d3.js Pie Chart Generator
     * Generates data from an array of data.
     */
    var pie = d3.layout.pie()
        .sort(null)
        .value(function(d) { return d.value; });

    var svg = d3.select("#chart").append("svg")
        .attr("width", width)
        .attr("height", height)
        .append("g")
        .attr("transform", "translate(" + width / 2 + "," + height / 2 + ")");

    // d3.json() calls the Rapido endpoint @@rapido/rating/search (a rest api␣
↪endpoint)
    d3.json("@@rapido/rating/search")
        // d3.json() puts the authenticator token in the X-Csrf-Token header,
        .header("X-Csrf-Token", authenticator)
        // and d3.json() passes the search query in the request BODY.
        .post(
            JSON.stringify({"query": "total>0"}),
            function(err, results) {
                var data = [];
                var color = d3.scale.linear().domain([0,results.length]).range(["
↪#005880","#9abdd6"]);
                var index = 0;
                results.forEach(function(d) {
                    var label = d.items.id.split('/')[d.items.id.split('/').length -␣
↪1];
```

(continues on next page)

(proviene de la página anterior)

```
                    data.push({
                        'i': index,
                        'value': d.items.total,
                        'label': label
                    });
                    index += 1;
                });

                // add arc element
                var g = svg.selectAll(".arc")
                    // call pie() function
                    .data(pie(data))
                    // add g element
                    .enter().append("g")
                    .attr("class", "arc");

                // add path element
                g.append("path")
                    .attr("d", arc)
                    .style("fill", function(d) { return color(d.data.i); });

                // add text element
                g.append("text")
                    .attr("transform", function(d) { return "translate(" + arc.
↪centroid(d) + ")"; })
                    .attr("dy", ".35em")
                    .style("text-anchor", "middle")
                    .text(function(d) { return d.data.label; })
                    .style("fill", "white");
            }
        );
});
```

Ese es un script bastante complejo, y no detallaremos aquí los aspectos relacionados con D3js (es sólo un ejemplo típico para dibujar un gráfico circular tipo pastel); Nos centraremos en la forma en que obtenemos los datos.

Lo primero que debe notar es la función `require`. Es una característica de la librería RequireJS (provista con Plone por defecto) para cargar nuestras dependencias.

Tenemos 2 dependencias:

- `mockup-utils`, que es un recurso interno de Plone,
- La librería D3js (y lo cargamos pasando su URL remota a RequireJS).

`mockup-utils` nos permite obtener el token de autenticador (con el método `getAuthenticator`), lo necesitamos para usar la API REST de Rapido.

Nota:

- RequireJS o `mockup-utils` no son obligatorios para usar el API REST de Rapido, si estuviéramos fuera de Plone (utilizando Rapido como backend remoto), nosotros deberíamos hacer una llamada a `../@@rapido/rating` que devuelve el token en una cabecera HTTP. Solo los utilizamos porque son proporcionados por Plone por defecto, y facilitan nuestro trabajo.
- En vez de la forma de cargar directamente D3 desde su CDN, podríamos haber puesto el archivo `d3.v3.min.js` en la carpeta `/rapido/rating`, y servirlo localmente.

La segunda parte interesante es la llamada al metodo `d3.json()`:

- ese llama al endpoint `@@rapido/rating/search`,
- ese pone el token de autenticador en el encabezado `X-Csrf-Token`,
- y pasa la consulta de búsqueda en la solicitud BODY.

Eso es básicamente lo que necesitamos hacer en cualquier framework JS que usamos (aquí usamos D3, pero podría ser un framework generalista como Angular, Backbone, Ember, etc.).

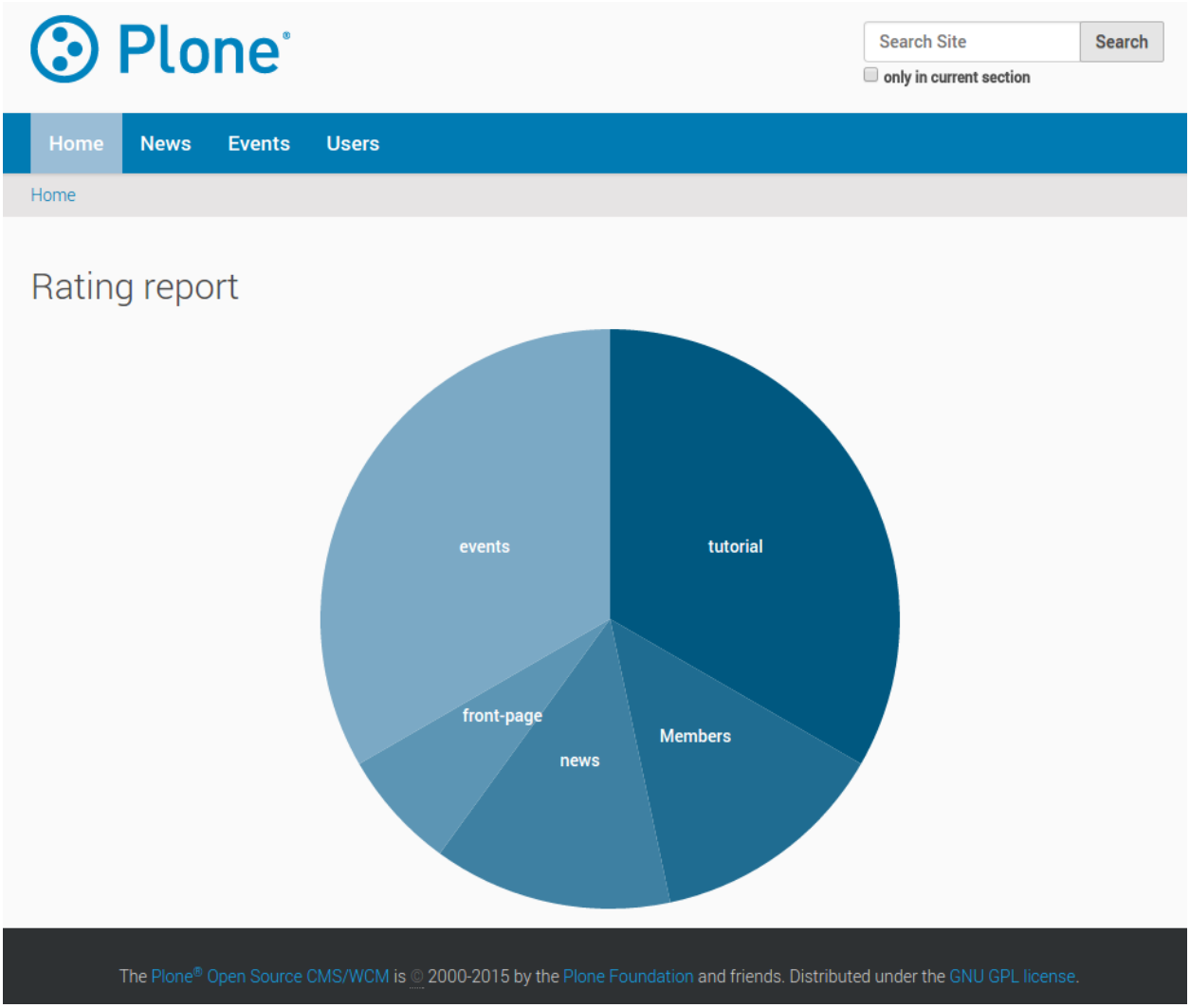
Ahora sólo necesitamos cargar este script desde nuestro bloque:

```
<h2>Rating report</h2>
<div id="chart"></div>
<script src="++theme++tutorial/rapido/rating/report.js"></script>
```

Y podemos visitarlo en la siguiente dirección URL:

http://localhost:8080/Plone/news/@@show-report

para ver un gráfico circular de pastel de los votos en todos los tipos de contenido *Noticias* en la sección del sitio llamada *News*!!!



Descargue los `códigos fuentes de este tutorial`.

Nota: Este archivo .zip se puede importar en el editor de temas, pero no se puede activar como un tema regular, ya que sólo contiene nuestra aplicación Rapido. La aplicación puede utilizarse desde nuestro tema principal añadiendo un archivo *rating.lnk* en la carpeta `rapido` nuestro tema actual, que contiene:

```
tutorial
```

indicando que la aplicación Rapido denominada `rating` se almacena en el tema denominado `tutorial`. Y luego podemos activar nuestras reglas específicas añadiendo:

```
<after css:content=".documentFirstHeading">
    <include css:content="form" href="/@@rapido/rating/blocks/rate" />
</after>

<rules css:if-content=".section-front-page">
    <before css:content=".documentFirstHeading">
        <include css:content="form" href="/@@rapido/rating/blocks/top5" />
    </before>
</rules>
```

en nuestro principal archivo `rules.xml` del tema Diazo.

2.5 Referencia

2.5.1 Aplicación

Una aplicación Rapido se define por una carpeta en la carpeta `rapido` del tema actual.

La carpeta de la aplicación puede contener un archivo `settings.yaml` en su raíz pero no es obligatorio. Permite definir los ajustes de control de acceso (véase *Control de acceso*) o habilitar el modo `debug`.

Siempre contiene una carpeta de `blocks` que contiene sus bloques (consulte *Bloques*).

También puede contener elementos regulares de un tema como(archivos `rules.xml`, CSS, Javascript, etc.).

Localización de una aplicación Rapido fuera del tema actual

Si utilizamos muchas aplicaciones Rapido, o si el tema y las aplicaciones Rapido son administradas por diferentes personas, puede ser preferible ubicar las aplicaciones Rapido en un tema dedicado.

Para ello, solo necesitamos referenciarlo utilizando un archivo de texto plano con extensión `.lnk` en el tema actual. El nombre de archivo debe ser el id de la aplicación y su contenido debe ser el ID del tema.

Por ejemplo, nuestro tema activo se estructuraría de la siguiente manera:

```
/rapido
    myapp.lnk
```

El archivo `myapp.lnk` sería solo:

```
dev-theme
```

El tema `dev-theme` podría contener la completa aplicación Rapido llamada `myapp`:

```
/rapido
    /myapp
        settings.yaml
        /blocks
            stats.html
```

Y todo funcionará como si la carpeta `myapp` estuviera en nuestro tema activo.

2.5.2 Bloques

Un bloque se define mediante 3 archivos almacenados en la carpeta `blocks` de la aplicación. Esos archivos tienen el mismo nombre de archivo (que es el identificador de bloque) con las extensiones `.html`, `.py` y `.yaml`.

El archivo HTML

El archivo `.html` contiene el diseño del bloque. Es HTML normal. Los elementos dinámicos están encerrados entre corchetes. Ejemplo:

```
<p>This is a dynamic message: {message}</p>
```

Los corchetes serán reemplazados por el valor del elemento correspondiente.

Si el elemento es un elemento BASIC y devuelve un objeto, podemos acceder a sus propiedades. Ejemplo:

```
<h1>{my_doc.title}</h1>
```

Del mismo modo, si un elemento BASIC devuelve un diccionario, podemos acceder a sus elementos. Ejemplo:

```
<p>{info[user]} said: {info[comment]}</p>
```

Cuando se procesa, el diseño del bloque se envuelve en un elemento HTML `<form>`.

El diseño puede contener el marcado de patrones de Mockup, se renderizará como se esperaba.

Es posible que algunos patrones de Mockup tengan que representar los corchetes reales en la salida. Doble ellos para escapar de ellos:

```
<a href="#modal" class="pat-plone-modal"
    data-pat-modal='{{"content": "form"}}'>Display modal</a>
```

Una vez procesado, si el bloque contiene algunos enlaces con un destino `ajax`:

```
<a href="@@rapido/record/1234" target="ajax">Open</a>
```

la solicitud se cargará en modo AJAX y su contenido reemplazará el contenido del bloque actual.

Plantilla TAL

La plantilla HTML sólo ofrece la inserción de elementos. Si necesitamos más características de plantilla, el archivo `.html` puede ser reemplazado por un archivo `.pt`, y podemos usar los comandos TAL.

En el contexto de una Page Template, los elementos de bloque están disponibles en el objeto `elements`:

```
def my_title(context):
    return "Chapter 1"
```

```
<h1 tal:content="elements/my_title"></h1>
```

Los elementos se pueden utilizar como condiciones:

```
def is_footer(context):
    return True
```

```
<footer tal:condition="elements/is_footer">My footer</footer>
```

Si un elemento devuelve un objeto iterable (lista, diccionario), podemos hacer un bucle:

```
def links(context):
    return [
        {'url': 'https://validator.w3.org/', 'title': 'Markup Validation Service'},
        {'url': 'https://www.w3.org/Style/CSS/', 'title': 'CSS'},
    ]
```

```
<ul>
    <li tal:repeat="link elements/links">
        <a tal:attributes="link/url"
            tal:content="link/title"></a>
    </li>
</ul>
```

El contexto Rapido actual está disponible en el objeto `context`:

```
<h1 tal:content="context/content/title"></h1>
```

El archivo YAML

El archivo `.yaml` contiene: - los configuraciones de elementos (ver más abajo),

- la opción `target`: si se establece en `ajax`, cualquier acción en el bloque que resulte en un envío de formulario no redirigirá la página actual, solo actualizará el contenido del bloque a través de una llamada AJAX,
- la `view_permission` para quién puede ver el bloque (consulte *Control de acceso*).

El archivo Python

El archivo `.py` contiene la implementación de cada elemento como una función de Python cuyo nombre es el identificador de elemento y toma `context` como parámetro.

2.5.3 Elementos

Declaración

Los elementos se deben declarar en el archivo YAML bajo la entrada de `elements`. Cada elemento es declarado por su identificador, y su definición es:

- una lista de parámetros, por ejemplo:

```
elements:
    do_something:
        type: ACTION
        label: Do something
```

- o simplemente una cadena, en cuyo caso Rapido asumirá que es el parámetro `type`, por ejemplo:

```
elements:
    message: BASIC
```

es equivalente a:

```
elements:
    message:
        type: BASIC
```

Tipos

Hay diferentes tipos de elementos (definidos por el parámetro `type`):

- `BASIC`: una pieza de HTML devuelta por su función de implementación.
- `ACTION`: un botón que ejecutará la función de implementación al hacer clic. Su etiqueta viene dada por el parámetro `label`.
- `TEXT`: un campo de entrada de texto.
- `NUMBER`: un campo de entrada numérico.
- `DATETIME`: un campo de entrada de fecha / hora.

Elementos de entrada

Los elementos de entrada (es decir, `TEXT`, `NUMBER`, o `DATETIME`) se pueden indexar como `field` o `text`. La indexación se indica utilizando el parámetro `index_type`.

De forma predeterminada, los elementos de entrada son editables pero también pueden tener un `mode` diferente:

- `COMPUTED_ON_SAVE`: el valor se calcula cada vez que se guarda el registro,
- `COMPUTED_ON_CREATION`: el valor se calcula cuando se crea el registro.

Elementos de acción

Los elementos de acción se representan como botones de envío y permiten activar una llamada a una función asociada de Python.

Si la función devuelve un valor, debe ser una cadena y se utilizará como dirección URL de redirección para la solicitud actual.

Esta es la forma de redirigir a otra ubicación una vez que se ha ejecutado la acción.

Acciones adicionales

Las siguientes acciones se pueden incluir en nuestro diseño HTML de bloque y no requieren una función asociada de Python:

- `_save`: crea un registro basado en los valores enviados de los elementos de campo y luego redirige a la visualización de registros en modo de lectura;
- `_edit`: abre el registro actual en modo de edición;
- `_delete`: elimina el registro actual.

Llamada HTTP directa a elementos

Normalmente queremos mostrar bloques, pero también podemos llamar a un elemento por su dirección URL:

http://localhost:8080/Plone/@@rapido/myapp/blocks/block1/element1

Se admiten las solicitudes GET y POST.

Si el elemento es una acción, se ejecutará su función Python; El valor devuelto se supone que es una cadena y se utilizará como una dirección URL de redirección. Al construir una aplicación, nos permite crear enlaces que redirigirán al usuario a la ubicación correcta dependiendo de nuestros criterios de negocio (por ejemplo, si el usuario pertenece al grupo A, vaya a la `page1`, o bien vaya a la página `page2`).

Si el elemento no es una acción, su función Python se ejecutará y el resultado se devolverá como una respuesta.

Nota: Podemos cambiar el tipo de contenido de respuesta como este:

```
def my_element(context):
    context.request.reponse.setHeader('content-type', 'text/csv')
    return "one,two,three\n1,2,3"
```

2.5.4 Registros

Los registros Rapido se pueden crear guardando un bloque que contiene elementos de campo. El valor de cada elemento enviado se almacenará en un elemento correspondiente.

En ese caso, el registro tiene un bloque asociado (el identificador de bloque se almacena en un elemento llamado `block`). Cuando el registro se representa para su visualización (cuando cargamos su dirección URL en nuestro navegador), utiliza el diseño del bloque con nombre.

Los registros también se pueden crear manualmente (sin ningún bloque asociado) utilizando la API de Python o la API REST. Tales registros no se pueden renderizar automáticamente llamando a su dirección URL, pero sus valores de elemento se pueden usar en un bloque si sabemos cómo encontrar el registro (en el *Tutorial* por ejemplo, nuestros registros se crean manualmente a partir de la función `like`, no están asociados Con el bloque de `rate`, pero usamos los elementos almacenados para producir nuestros contenidos de elementos).

2.5.5 Funciones asociadas de Python

Para un elemento `BASIC`, la función Python asociada (que tiene el mismo id) devolverá el contenido del elemento.

Para los elementos de campo (`TEXT`, `NUMBER`, `DATETIME`), la función Python asociada devolverá su valor predeterminado.

Para un elemento `ACTION`, la función Python asociada se ejecutará cuando se active la acción.

Funciones especiales de Python

on_save Se ejecuta cuando se guarda un registro con el bloque. Si devuelve un valor, debe ser una cadena y se utilizará como dirección URL de redirección para la solicitud actual.

on_display Se ejecuta cuando se muestra un bloque. Se ejecutará antes de todas las funciones del elemento. Se puede utilizar para hacer algún cálculo y poner el resultado en el `context` para que pueda ser accedido por los diferentes elementos. También se puede usar para redireccionar a otra página (usando `context.request.response.redirect()`).

on_delete Se ejecuta cuando se elimina un registro. Si devuelve un valor, debe ser una cadena y se utilizará como dirección URL de redirección para la solicitud actual.

record_id Se ejecuta en el momento de la creación para calcular el id del registro.

2.5.6 Indexación y búsqueda

El sistema de almacenamiento Rapido (souper) soporta la indexación.

Cualquier elemento de bloque se puede indexar agregando una configuración `index_type` en su definición YAML.

El ajuste `index_type` puede tener dos valores posibles:

- `field`: estos índices coinciden con los valores exactos y admiten consultas de comparación, consultas de rangos y clasificación.
- `text`: dicho índice corresponde a palabras contenidas (aplicable sólo a valores de texto).

Las consultas utilizan el formato (CQE format).

Ejemplo (asumiendo que el `author`, `title` y `price` son índices existentes):

```
context.app.search(
    "author == 'Conrad' and 'Lord Jim' in title",
    sort_index="price")
```

Los registros se indexan en el momento en que se guardan. Podemos forzar la reindexión mediante la API de Python:

```
myrecord.reindex()
```

También podemos reindexar todos los registros usando el comando URL `refresh`:

http://localhost:8080/Plone/@@rapido/<app-id>/refresh?_authenticator=<valid token>

o usando la API REST (ver *API REST*).

2.5.7 Mostrando Rapido en Plone

Podemos ver un bloque visitando su dirección URL:

http://localhost:8080/Plone/@@rapido/myapp/blocks/simpleblock

Del mismo modo para un registro:

http://localhost:8080/Plone/@@rapido/myapp/record/my-record-id

Pero simplemente devuelve el HTML generado por el bloque.

Para mostrar nuestros bloques en nuestro sitio de Plone, hay 4 posibilidades:

Reglas diazo

Podemos utilizar la directiva Diazo `include` para obtener el contenido del bloque Rapido e inyectarlo en nuestras páginas con las directivas `before`, `after` o `replace`.

Ejemplo:

```
<before css:content="#content-core">
    <include css:content="form" href="/@@rapido/myapp/blocks/simpleblock" />
</before>
```

Vistas extra

Advertencia: Desde rapido.plone 1.1, podemos declarar vistas de primera clase de Plone desde Rapido, pero requiere plone.resources 1.2.

Si no queremos simplemente inyectar un pequeño fragmento de HTML en las páginas existentes, sino usar un bloque Rapido como parte principal de la página, podemos declarar un bloque como una vista en su archivo YAML:

```
view:
    id: my-custom-view
    with_theme: true
```

Y luego llamamos a `@@my-custom-view` en cualquier contenido, como:

http://localhost:8080/Plone/news/@@my-custom-view

y muestra nuestro bloque como contenido de la página principal.

Si la `with_theme` es `false`, la página se procesa sin el tema Plone (solo obtenemos el bloque por sí mismo).

Vistas extra antes de 1.1

DESCONTINUADA desde rapido.plone 1.1

Si no queremos simplemente inyectar un pequeño fragmento de HTML en las páginas existentes, sino crear una nueva vista para nuestros contenidos, podemos usar las **vistas neutras** de Rapido.

Las vistas neutrales se obtienen agregando `@@rapido/view/<any-name>` a una dirección URL de contenido. Simplemente devolverá la vista predeterminada del contenido (por eso los llamamos neutrales).

Por ejemplo, todas esas direcciones URL muestran lo mismo:

```
http://localhost:8080/Plone/front-page
http://localhost:8080/Plone/front-page/@@rapido/view/
http://localhost:8080/Plone/front-page/@@rapido/view/abc
http://localhost:8080/Plone/front-page/@@rapido/view/123
```

Así que podemos llamar a un contenido con una URL que controlamos, y eso nos permite crear reglas específicas de Diazo para ello usando el atributo `if-path`.

Inyección codificada

```
<rules if-path="@@rapido/view/show-report">
    <replace css:content="#content">
        <include css:content="form" href="/@@rapido/stats/blocks/report" />
    </replace>
</rules>
```

En este ejemplo, si abrimos la siguiente dirección URL:

http://localhost:8080/Plone/@@rapido/view/show-report

veremos el contenido principal de nuestra página es reemplazado por nuestro bloque de `report`.

Inyección dinámica

También podemos mostrar dinámicamente un recurso Rapido especificado en la URL. Rapido proporciona un patrón de inyección de URL que permite referirse a la solicitud de los padres en nuestra regla Diazo.

El patrón es: `$<integer>`, donde el entero especifica la posición inicial después de `@@rapido` para obtener la ruta de inyección.

Por ejemplo:

- con `http://localhost:8080/Plone/@@rapido/view/show-report/5654654`, `$3` recibe la parte de la ruta a partir del tercer elemento después de `@@rapido`, el cual es: `5654654`,
- con `http://localhost:8080/Plone/@@rapido/view/show-report/myapp/record/5654654`, `$3` recibe la parte de la ruta que comienza en el tercer elemento después de `@@rapido`, el cual es: `myapp/record/5654654`,
- con `http://localhost:8080/Plone/@@rapido/view/show-report/myapp/record/5654654/edit`, `$5` obtiene la parte de la ruta que comienza en el quinto elemento después de `@@rapido`, el cual es: `5654654/edit`.

Ejemplos:

```
<rules if-path="@@rapido/view/show-report">
    <replace css:content="#content-core">
        <include css:content="form" href="/@@rapido/$3" />
    </replace>
</rules>
```

si abrimos la siguiente dirección URL:

http://localhost:8080/Plone/@@rapido/view/show-report/myapp/record/my-record-id

nos representaría `myapp/record/my-record-id` en nuestro de la página principal de contenido.

También podríamos hacer:

```
<rules if-path="@@rapido/view/show-report">
    <replace css:content="#content-core">
        <include css:content="form" href="/@@rapido/myapp/record/$3" />
    </replace>
</rules>
```

si abrimos la siguiente dirección URL:

http://localhost:8080/Plone/@@rapido/view/show-report/my-record-id

obtendremos la misma representación que en nuestro ejemplo anterior.

Mosaic

Mosaic es un editor de diseño.

Permite agregar y manipular *tiles* en nuestros diseños de contenido.

Rapido proporciona un tile de mosaico, así que cualquier bloque de Rapido se puede agregar como tile a nuestras disposiciones del diseño.

Para habilitarlo, necesitamos instalar Mosaic y luego importar un perfil específico de Rapido Generic Setup llamado **«rapido.plone mosaic tile»** desde el *ZMI* >>> *portal_setup* >>> *Import* y haga clic en el botón **«Import all steps»**.

Aquí el enlace de la pestaña **«Import»** desde la herramienta **portal_setup** para ejecutar el perfil Generic Setup:

> http://localhost:8080/Plone/portal_setup/manage_fullImport

Patrones Mockup

Algunos patrones de Mockup pueden mostrar contenido proporcionado por una URL. Los dos principales casos de uso son:

- **Mostrar un bloque Rapido en un modal**: usamos el patrón `plone-modal` en un elemento `<a>`, la URL del bloque Rapido será proporcionada en su atributo `href`, y solo necesitamos especificar `form.rapido-block` como selector de contenido (porque `plone-modal` el selector de contenido por defecto es `#content`, el cual es preciso para una página de Plone pero no para un bloque de Rapido). Ejemplo:

  Creamos un bloque llamado `my-content` contiene lo que sea necesario, y creamos un bloque llamado `menu` contiene el siguiente HTML:

```
<a href="@@rapido/my-app/blocks/my-content"
    class="plone-btn pat-plone-modal"
    data-pat-plone-modal="content: form.rapido-block">
        Open in a modal
</a>
```

  Y entonces solo necesitamos insertar `menu` en nuestra página de Plone (usando una regla de Diazo).

  Consulte la documentación modal de Mockup para obtener más detalles sobre las opciones.

- **Cargar un bloque Rapido dinámicamente en la página actual**: usamos el `plone-contentloader` para inyectar nuestro bloque Rapido donde queramos. En nuestro ejemplo anterior, cambiaríamos el `menu` HTML a:

```
<a href="#" class="pat-contentloader"
   data-pat-contentloader="url:@@rapido/my-app/blocks/my-content#form.rapido-
↪block;">
   Load content</a>
```

  Reemplazaría el enlace «Cargar contenido» con nuestro bloque `my-content` cuando haga clic en el enlace.

  > **Advertencia:** con `plone-contentloader`, el selector de contenido se pasa directamente como un hash al final de la URL.

  `plone-contentloader` también nos permite orientar un elemento específico para la inyección (en lugar de reemplazar el enlace):

```
<a href="#" class="pat-contentloader"
   data-pat-contentloader="url:@@rapido/my-app/blocks/my-content#form.rapido-
↪block;target:#here;">
   Load content</a>
<p id="here">Insert my content here.</p>
```

De forma predeterminada, la inyección se activa con un clic, pero podemos elegir cualquier evento de DOM (`mouseover` por ejemplo), e incluso podemos realizar la inyección inmediatamente (usando el trigger `immediate`):

```
<a href="#" class="pat-contentloader"
   data-pat-contentloader="url:@@rapido/my-app/blocks/my-content#form.rapido-
↪block;trigger:immediate">
   Load content</a>
```

2.5.8 Javascript

Podemos agregar archivos Javascript a nuestro tema que interactuará con nuestros bloques Rapido.

No hay restricciones específicas en estos scripts. Sin embargo, podría ser útil utilizar las dependencias Javascript ya proporcionadas por Plone, como `jQuery` y `require`.

Como Rapido permite cargar dinámicamente el contenido del bloque (usando el modo `ajax`), es posible que tengamos que saber cuándo se ha cargado dinámicamente un bloque Rapido.

Para ello podemos usar el evento `rapidoLoad`, que recibe el identificador de bloque como parámetro. Ejemplo:

```
require(['jquery'], function($) {
    $(document).on('rapidoLoad', function(event, block_id) {
        console.log(block_id + ' has been loaded!');
    });
});
```

2.5.9 Importar / exportar y gestión de código fuente

Las aplicaciones Rapido se implementan en la carpeta */rapido* de un tema Diazo. Todos los procedimientos de desarrollo conocidos para el tema se aplican al desarrollo Rapido.

Importar / exportar ZIP

El editor de temas de Plone permite exportar un tema Diazo como un archivo ZIP o importar un nuevo tema desde un archivo ZIP.

Esa es la forma en que vamos a importar / exportar nuestras aplicaciones Rapido entre nuestros sitios.

Edición de código fuente directamente

También podríamos almacenar nuestros temas Diazo en nuestro servidor en la carpeta de instalación de Plone:

```
$INSTALL_FOLDER/resources/theme/my-theme
```

De esta manera, podemos desarrollar nuestras aplicaciones Rapido utilizando nuestras herramientas habituales de desarrollo (editor de texto o IDE, Git, etc.).

Complemento de Plone

También podemos crear nuestro propio complemento de Plone (consulte la documentación de Plone, y formación de Plone) y gestionar nuestras aplicaciones de Rapido en su carpeta de temas.

2.5.10 Control de acceso

Lista de control de acceso

La ACL definida en la aplicación se aplica a registros, no a bloques. Los bloques siempre son accesibles, si no queremos que un bloque proporcione alguna información, tenemos que implementar esto en su archivo Python o usar la configuración `view_permission`.

Además, el control de acceso sólo afecta el acceso HTTP directo a los registros (como abrir una dirección URL de registro, eliminar un registro a través de la API de JSON, etc.) y no afecta lo que sucede en los archivos de Python de bloque.

Por ejemplo, en el *Tutorial*, si un visitante anónimo hace clic en el botón *«Like»* de una página en la que nadie había votado todavía, la función `like` creará un registro.

Pero un visitante anónimo no podría modificar este registro o eliminarlo usando la API de JSON.

El formato esperado es:

```
acl:
  rights:
    reader: [<list of users or groups>]
    author: [<list of users or groups>]
    editor: [<list of users or groups>]
  roles: {<role_id>: [<list of users or groups>]}
```

En la lista de usuarios o grupos, `'*'` significa todo el mundo.

Niveles de acceso

Los niveles de acceso son:

- `reader`: puede leer todos los registros,
- `author`: puede leer todos los registros, puede crear registros, puede modificar / borrar sus propios registros,
- `editor`: puede leer / modificar / borrar cualquier registro, puede crear registros.

El control de acceso se gestiona en el archivo `settings.yaml` de la carpeta raíz de la aplicación.

Roles

Los roles no otorgan derechos específicos en los registros, pueden definirse libremente. Se utilizan en nuestras funciones de Python para cambiar el comportamiento de la aplicación dependiendo del usuario.

Por ejemplo, podríamos tener un rol llamado “PurchaseManager”, y en nuestro bloque mostraríamos un botón «Validate purchase» si el usuario actual tiene el rol “PurchaseManager”.

Permisos sobre bloques

De forma predeterminada, los bloques son accesibles por cualquier persona (incluidos los visitantes anónimos).

Al configurar el atributo `view_permission` en el archivo YAML de un bloque, podemos controlar el acceso a este bloque.

Su valor es una lista de usuarios o grupos.

Ejemplo:

```
elements:
  whatever: BASIC
view_permission:
  PurchaseDepartment
  eric
```

Este bloque será accesible sólo por los miembros del grupo "PurchaseDepartment" y Eric.

Esta restricción se aplica a la representación directa de bloques y las llamadas de elementos, incluidas las llamadas REST.

2.5.11 Reglas de contenido

Las reglas de contenido permiten activar acciones específicas (por ejemplo, enviar un mensaje de correo electrónico) cuando un evento determinado (por ejemplo, cuando se crea un nuevo contenido en tal y cual carpeta) que ocurra en nuestro sitio Plone.

Rapido proporciona una acción de regla de contenido, por lo que podemos ejecutar una función Rapido cuando sucede un evento dado.

La acción a tomar se define en el editor de reglas de contenido de Plone (consulte la documentación de las reglas de contenido de Plone), y requiere los siguientes parámetros:

- el id de la aplicación,
- el identificador de bloque,
- el nombre de la función.

El `context.content` recibido por la función será el contenido donde ocurrió el evento.

2.5.12 Llamada externa a Rapido

Al recorrer a `@@rapido-call`, podemos llamar a un elemento Rapido como una función Python.

Podría ser muy útil cuando queramos usar Rapido desde un PythonScript, una page template de Plone o cualquier mecanismo Plone que ofrezca ejecutar un pequeño script (flujo de trabajo Plone, `collective.easyform`, etc.).

`@@rapido-call` acepta los siguientes parámetros:

- `path` (obligatorio, cadena): Rapido ruta al elemento a llamar (formato: `app/blocks/element`),
- `content` (opcional, objeto): el contenido a proporcionar al contexto de Rapido,
- cualquier otro parámetro nombrado: los parámetros nombrados estarán disponibles para la implementación de Python del elemento en el diccionario `context.params`.

Ejemplo:

PythonScript:

```
visitors = container.restrictedTraverse('@@rapido-call')(
    'myapp/stats/analyse',
    content=portal.news,
    min_duration=2,
    client='smartphone')
```

Elemento Rapido en `myapp/stats.py`:

```
def analyse(context):
    filtered = get_filtered_visitors(
        duration=context.params['min_duration'],
        type=context.params['client'])
    return len(filtered)
```

2.6 API de Python

Cualquier función Python de Rapido recibe el `context` como parámetro.

El `context` proporciona las siguientes propiedades:

2.6.1 context.app

Esta propiedad da acceso al objeto de aplicación Rapido.

Propiedades

acl Devuelve el objeto de la lista de control de acceso de la aplicación Rapido (ver abajo).

blocks Devuelve los identificadores de bloque existentes.

indexes Devuelve los identificadores de índice existentes.

url Devuelve la URL de la aplicación.

Métodos

create_record(self, id=None) Crea y devuelve un nuevo registro. Si no se proporciona `id`, se genera una predeterminada. Si `id` ya existe, se sustituye por otro (como `...-1`, `...-2`).

delete_record(self, id=None, record=None, ondelete=True) Elimine el registro (que se puede pasar como objeto o como id). Si la `ondelete` es verdad, la función `on_delete` será llamada (si existe) antes de borrar el registro.

get_block(self, block_id) Devuelve un bloque.

get_record(self, id) Devuelve el registro correspondiente al `id`, o `None` si no existe.

log(self, message) Registra un mensaje en el registro del servidor. Si la aplicación está en modo de depuración, registra el mismo mensaje en la consola javascript del navegador. Los mensajes pueden ser cadenas o cualquier otro objeto serializable.

records(self) Devuelve todos los registros como una lista.

_records(self) Devuelve todos los registros como un generador de Python.

search(self, query, sort_index=None, reverse=False) Realiza una búsqueda y devuelve registros como una lista.

_search(self, query, sort_index=None, reverse=False) Realiza una búsqueda y devuelve registros como un generador de Python.

2.6.2 `context.request` and `context.parent_request`

`context.request` es la actual request a Rapido, como:

> http://localhost:8080/Plone/@@rapido/rating/blocks/rate

Cuando un bloque está embebido en una página de Plone, `context.request` fue emitido por el navegador del usuario, fue emitido por Diazo.

Para obtener el request emitida por el navegador del usuario, nosotros usamos `context.parent_request`.

Ambos son objetos de solicitud HTTP, consulte la documentación de referencia.

Ejemplos:

- Lectura de valores presentados:

```
val1 = context.request.get('field_1')  # returns None if key doesn't exist
val1 = context.request['field_2']  # fail if key doesn't exist
```

- Lectura del `BODY`:

```
request.get('BODY')
```

2.6.3 `context.portal`

Devolver el objeto de portal Plone.

Es equivalente a:

```
context.api.portal.get()
```

La tarea más común que realizaremos a través del objeto de portal es obtener su contenido:

```
folder = context.portal['my-folder']
```

2.6.4 context.content

Devuelve el contenido actual de Plone.

Las tareas más comunes que realizaremos en el contenido son:

- leer / escribir sus atributos (lectura / escritura):

```
the_tile = context.content.title
context.content.title = "I prefer another title"
```

- obteniendo su URL:

```
context.content.absolute_url()
```

Para manipular el contenido, consulte la documentación de la API de Plone sobre el contenido.

Nota: Dependiendo de su tipo de contenido, el objeto de contenido puede tener métodos y propiedades muy diferentes.

2.6.5 context.record

Devuelve el registro Rapido actual si lo hay.

Consulte *Registro* para obtener más información.

2.6.6 context.api

Da acceso completo a la API de Plone.

Advertencia: No es necesario importar la API, como se muestra en todos los ejemplos de API de Plone:

```
from plone import api  # WRONG
```

porque la API ya está disponible en el *context* de Rapido:

```
catalog = context.api.portal.get().portal_catalog
```

Esta API permite principalmente:

- buscar contenidos; por ejemplo:

```
folders = context.api.content.find(portal_type="Folder")
# be careful, the find() method returns Brain objects, not real objects
# so only indexed attributes are available.
desc = folders[0].Description # OK
folders[0].objectIds() # WRONG!
folder = folders[0].getObject()
folder.objectIds() # OK!
```

- para manipular contenidos (create / delete / move / publish / etc), ejemplo:

```
new_page = context.api.content.create(
    type='Document',
    title='My Content',
    container=context.content)
context.api.content.transition(obj=new_page, transition='publish')
```

- para acceder o gestionar la información de usuarios y grupos y enviar correos electrónicos. Ejemplo:

```
current_user = context.api.user.get_current()
context.api.portal.send_email(
    recipient=current_user.getProperty("email"),
    sender="noreply@plone.org",
    subject="Hello",
    body="World",
)
```

Para ejemplos más detallados , consulte la documentación de la API de Plone.

2.6.7 `context.rapido`

`context.rapido` es una función capaz de obtener otra aplicación Rapido en nuestro script actual.

Toma como parámetro obligatorio el id de la aplicación Rapido. Ejemplo:

```
purchase_app = context.rapido('purchase')
new_purchase_order = purchase_app.create_record()
```

También puede aceptar un parámetro `content` para proporcionar un específico contexto contenido a la aplicación (si no se proporciona, se tomará el contenido actual). Ejemplo:

```
stat_app = context.rapido('stats', content=context.portal.news)
```

2.6.8 `context.modules`

Advertencia: Por razones de seguridad, no se permite importar un módulo Python en un archivo Rapido Python.

Rapido proporciona algunos módulos seguros a través de `context.modules`:

- `context.modules.datetime`: Tipos básicos de fecha y hora,
- `context.modules.random`: Genera números pseudo-aleatorios,
- `context.modules.time`: Tiempo de acceso y conversiones.

Si necesitamos agregar módulos a `context.modules`, podemos hacerlo agregando en nuestro propio complemento de la siguiente forma:

```
import re
from rapido.core import app

app.safe_modules.re = re
```

En este ejemplo, permitimos acceder a `context.modules.re` desde nuestros archivos Python Rapido.

2.6.9 Registro

Propiedades

url Devuelve la URL del registro.

id Devuelve el identificador de registro.

Métodos

display(self, edit=False) Representa el registro utilizando su bloque asociado (si existe).

get(self, name, default=None) Devuelve el valor del elemento (y el valor predeterminado es el `default` si el elemento no existe).

items(self) Devuelve todos los elementos almacenados.

reindex(self) Vuelva a indexar el registro.

save(self, request=None, block=None, block_id=None, creation=False) Actualizar el registro con los elementos proporcionados e indexarlo.

`request` puede ser una request actual HTTP o un diccionario.

Si se menciona un bloque, se ejecutarán fórmulas (`on_save`, elementos calculados, etc.).

Si no hay bloque (y `request` es un diccionario), solo salvamos los valores de los elementos.

set_block(self, block_id) Asigne un bloque al registro. El bloque se utilizará entonces para representar el registro o para guardarlo.

Interfaz de diccionario de Python

Los elementos del registro se pueden acceder y manipular como elementos del diccionario:

```
myrecord['fruit'] = "banana"
for key in myrecord:
    context.app.log(myrecord[key])
if 'vegetable' in myrecord:
    del myrecord['fruit']
```

Nota: Cuando se establece un valor de elemento, el registro no se reindexa.

2.6.10 Lista de control de acceso

Nota: La lista de control de acceso a la aplicación puede ser obtenida mediante `context.app.acl`.

Métodos

current_user(self) Devuelve el ID de usuario actual. Equivalente a:

```
context.api.user.get_current().getUserName()
```

current_user_groups(self) Devuelve los grupos al usuario actual al que pertenece. Equivalente a:

```
api.user.get_current().getGroups()
```

has_access_right(self, access_right) Devuelve `True` si el usuario actual tiene el derecho de acceso especificado (los derechos de acceso Rapido son `reader`, `author`, `editor`, `manager`)

has_role(self, role_id) Devuelve `True` si el usuario actual tiene la función especificada.

roles(self) Devuelve los roles existentes.

2.7 API REST

2.7.1 Obtén las configuraciones de la aplicación

Request

```
GET /:site_id/@@rapido/:app_id
Accept: application/json
```

Response

```
{"no_settings": {}}
```

Cabecera Response HTTP

```
x-csrf-token: token
```

Devuelve la configuración de la aplicación Rapido y establece un token en el valor del encabezado HTTP `X-CSRF-TOKEN`.

Esta cabecera HTTP tendrá que ser reutilizada en todas las solicitudes hechas a la API (excepto para las solicitudes GET).

2.7.2 Autenticación

Algunas de las operaciones siguientes requieren autenticación antes de que se ejecuten correctamente. Tendrá que generar una cadena de autorización (una versión codificada Base64 de su nombre de usuario y contraseña separados por un punto).

Cadena Básica de Autenticación

Si su nombre de usuario es «john» y su contraseña es «password», puede generar rápidamente la cadena de autorización básica en el indicador python de la siguiente manera:

```
>>> "john.password".encode('base64','strict').strip()
'am9obi5wYXNzd29yZA=='
```

Ahora usted usa esta cabecera en todos sus requests:

```
Authorization: Basic am9obi5wYXNzd29yZA==
```

Nota: El X-CSRF-TOKEN esperado cambiará cuando utilice un encabezado de autorización básica.

2.7.3 Calcular un elemento

Request

```
GET /:site_id/@@rapido/:app_id/blocks/:block_id/:element_id
Accept: application/json
X-CSRF-TOKEN: :token
```

o:

```
POST /:site_id/@@rapido/:app_id/blocks/:block_id/:element_id
Accept: application/json
X-CSRF-TOKEN: :token
```

Response

```
{"something": "bla"}
```

Devuelve el valor devuelto por el cálculo del elemento. El X-CSRF-TOKEN no es necesario para un GET si el cálculo no produce ningún cambio.

2.7.4 Obtener un registro

Request

```
GET /:site_id/@@rapido/:app_id/record/:record_id
Accept: application/json
X-CSRF-TOKEN: :token
```

Response

```
{"item1": "value1", "id": "boom"}
```

Devuelve los elementos registro.

2.7.5 Obtiene todos los registros

Request

```
GET /:site_id/@@rapido/:app_id/records
Accept: application/json
X-CSRF-TOKEN: :token
```

Response

```
[{"path": "http://localhost:8080/demo/@@rapido/test2/record/boom", "id": "boom",
↪"items": {"bla": "bla", "id": "boom"}},
 {"path": "http://localhost:8080/demo/@@rapido/test2/record/10025657", "id": "10025657
↪", "items": {"id": "10025657"}},
 {"path": "http://localhost:8080/demo/@@rapido/test2/record/9755269", "id": "9755269",
↪ "items": {"bla": "bli", "id": "9755269"}},
 {"path": "http://localhost:8080/demo/@@rapido/test2/record/8742197835653", "id":
↪"8742197835653", "items": {"bla": "bli", "id": "8742197835653"}},
 {"path": "http://localhost:8080/demo/@@rapido/test2/record/9755345", "id": "9755345",
↪ "items": {"id": "9755345"}}]
```

Devuelve todos los registros.

2.7.6 Crear un nuevo registro

Request

```
POST /:site_id/@@rapido/:app_id
Accept: application/json
X-CSRF-TOKEN: :token
{"item1": "value1"}
```

Response

```
{"path": "http://localhost:8080/demo/@@rapido/test2/record/9755269", "id": "9755269",
↪"success": "created"}
```

Crear un nuevo registro con los elementos proveídos.

2.7.7 Crear muchos registros

Request

```
POST /:site_id/@@rapido/:app_id/records
Accept: application/json
X-CSRF-TOKEN: :token
[{"item1": "a"}, {"item1": "b", "item2": "c"}]
```

Response

```
{"total": 2, "success": "created"}
```

Creación masiva de registros.

2.7.8 Crear un nuevo registro por id

Request

```
PUT /:site_id/@@rapido/:app_id/record/:record_id
Accept: application/json
X-CSRF-TOKEN: :token
{"item1": "value1"}
```

Response

```
{"path": "http://localhost:8080/demo/@@rapido/test2/record/boom", "id": "boom",
↪"success": "created"}
```

Crear un nuevo registro con los elementos proveídos y teniendo el id especifico.

2.7.9 Elimina un registro

Request

```
DELETE /:site_id/@@rapido/:app_id/record/:record_id
Accept: application/json
X-CSRF-TOKEN: :token
```

Response

```
{"success": "deleted"}
```

Elimina el registro.

2.7.10 Remueve todos los registros

Request

```
DELETE /:site_id/@@rapido/:app_id/records
Accept: application/json
X-CSRF-TOKEN: :token
```

Response

```
{"success": "deleted"}
```

Remueve todos los registros y elimina los índices.

2.7.11 Actualiza un registro

Request

```
POST /:site_id/@@rapido/:app_id/record/:record_id
Accept: application/json
X-CSRF-TOKEN: :token
{"item1": "newvalue1"}
```

o:

```
PATCH /:site_id/@@rapido/:app_id/record/:record_id
Accept: application/json
X-CSRF-TOKEN: :token
{"item1": "newvalue1"}
```

Response

```
{"success": "updated"}
```

Actualiza el registro con los elementos proveídos.

2.7.12 Búsqueda de registros

Request

```
POST /:site_id/@@rapido/:app_id/search
Accept: application/json
X-CSRF-TOKEN: :token
{"query": "total>0", "sort_index": "total"}
```

Response

```
[{"path": "http://localhost:8080/tutorial/@@rapido/rating/record//tutorial/news", "id
↪": "/tutorial/news", "items": {"total": 5, "id": "/tutorial/news"}},
 {"path": "http://localhost:8080/tutorial/@@rapido/rating/record//tutorial", "id": "/
↪tutorial", "items": {"total": 8, "id": "/tutorial"}}]
```

Búsqueda de registros.

2.7.13 Reíndice

Request

```
POST /:site_id/@@rapido/:app_id/refresh
Accept: application/json
X-CSRF-TOKEN: :token
```

Response

```
{"success": "refresh", "indexes": ["id", "total"]}
```

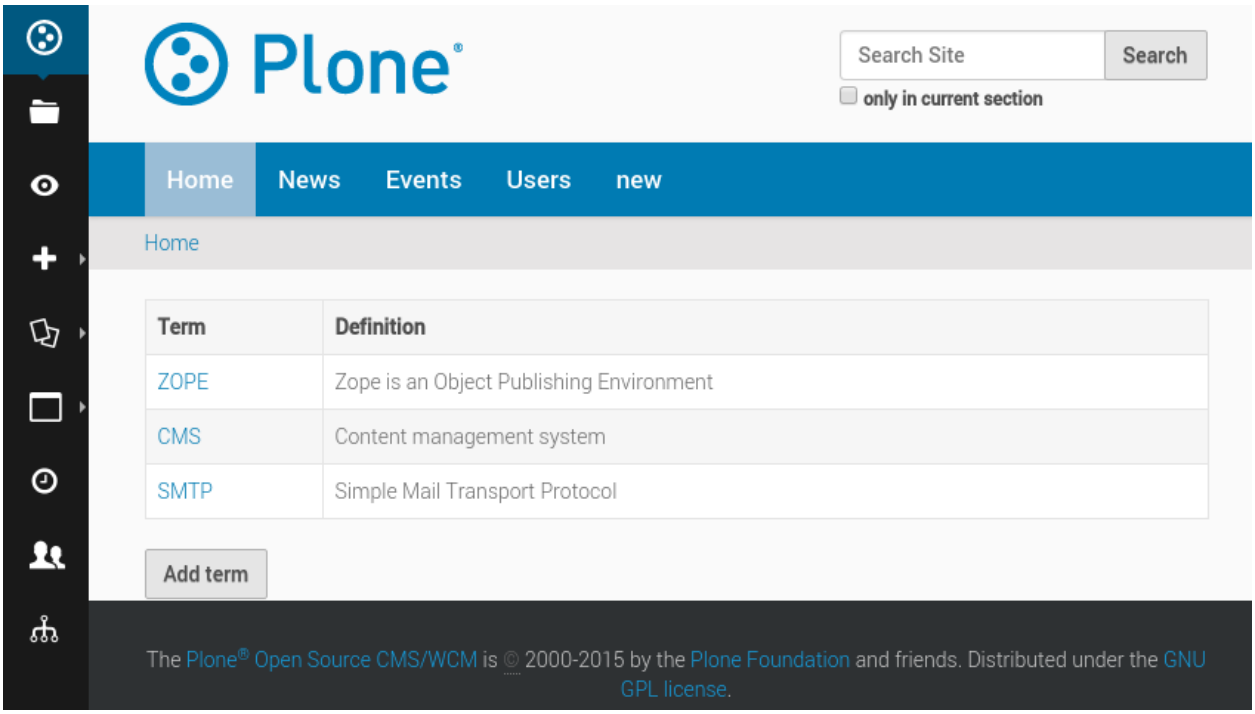
Vuelve a declarar los índices y vuelve a indexar todos los registros.

2.8 Casos de uso

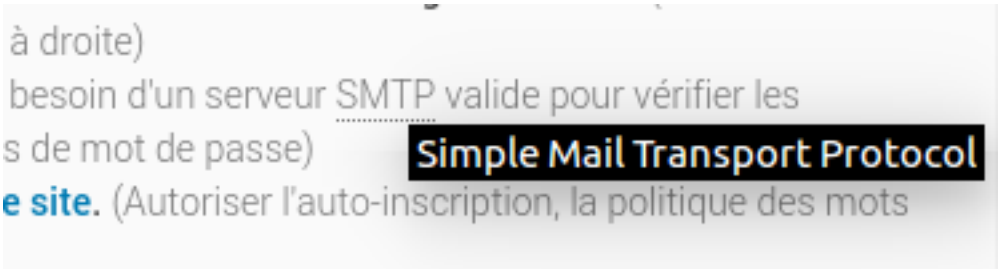
2.8.1 Caso de uso del glosario

Objetivo

Queremos proporcionar una herramienta para administrar una lista de términos y sus definiciones:



Cada vez que uno de estos términos aparezca en una página de nuestro sitio, será envuelto en una etiqueta *<abbr>*, cuyo título será la definición, por lo que cuando pasamos un término, obtenemos un pequeña mensaje emergente que indica su definición:



Estructura de la aplicación

```
rapido/
    glossary/
        blocks/
            all.html
            all.py
            all.yaml
            term.html
            term.py
            term.yaml
        glossary.js
rules.xml
```

El archivo rules.xml

```
<after css:theme-children="body">
  <script src="/tutorial/++theme++test/rapido/glossary/glossary.js"></script>
</after>
```

Esta regla inserta en todas nuestras páginas un archivo javascript a cargo de reemplazar las palabras coincidentes con etiquetas `<abbr>`.

El bloque `term`

Este bloque es un formulario que permite crear/editar/eliminar un término del glosario. Contiene dos elementos de campo y tres acciones.

- `term.html`

```
<p><label>Term</label> {term}</p>
<p><label>Definition</label> {definition}</p>
{_save} {_delete} {close}
```

- `term.yaml`

```
target: ajax
elements:
    term: TEXT
    definition: TEXT
```

(continues on next page)

(proviene de la página anterior)

```
    close:
        type: ACTION
        label: Close
    _save:
        type: ACTION
        label: Save
    _delete:
        type: ACTION
        label: Delete
```

- `term.py`

```
def close(context):
    return context.app.get_block('all').url

def on_save(context):
    return context.app.get_block('all').url

def on_delete(context):
    return context.app.get_block('all').url
```

Si hacemos clic en cualquier acción en este bloque, queremos ser redirigidos a la página de administración principal. Lo hacemos devolviendo la URL de `all` bloques (cuando una acción devuelve una cadena, se utiliza como una dirección URL de redirección).

El bloque `all`

Este bloque lista todos los términos existentes en una tabla. Cuando hacemos clic en un término, lo abrimos en el bloque `term` en modo de edición, y un botón permite abrir un bloque `term` en blanco para crear un nuevo término.

- `all.html`

```
<table class="listing"><tr><th>Term</th><th>Definition</th></tr>
{list}
</table>
{new_term}
```

- `all.yaml`

```
target: ajax
view:
    id: glossary
    with_theme: true
elements:
    list: BASIC
    new_term:
        type: ACTION
        label: Add term
```

La configuración `view` permite renderizar el bloque `all` como la vista de Plone llamada `@@glossary`, por lo que podemos llamar a http://localhost:8080/Plone/@@glossary para verlo.

- `all.py`

```
def list(context):
    html = u""
```

(continues on next page)

(proviene de la página anterior)

```
    for record in context.app.records():
        html += """<tr><td><a href="%s/edit" target="ajax">%s</a></td><td>%s</td>
↪</tr>""" % (
            record.url,
            record['term'],
            record['definition'],
        )
    return html

def new_term(context):
    return context.app.get_block('term').url
```

La función `list` crea una fila de tabla para cada registro existente, mostrando el valor del término y el valor de la *definición*. El enlace que ponemos en el término se dirige a la URL de registro (más */edit* para abrirlo en modo de edición) y hemos añadido *target=»ajax»* por lo que la página resultante no se muestra como una página completa, se acaba de cargar en la actual bloque en modo AJAX.

El Javascript

- `glossary.js`

```
require(['jquery'], function($) {
    if($('.template-edit').length > 0) {
        return
    }
    $.getJSON('/tutorial/@@rapido/glossary/records', function(data) {
        var keys = [];
        var values = {};
        for(var i=0; i<data.length; i++) {
            term = data[i].items.term;
            definition = data[i].items.definition;
            keys.push(term);
            values[term] = definition;
        }
        var re = RegExp("(\\W)(" + keys.join("|") + ")(\\W)", "g");
        function replaceNodeText() {
         if (this.nodeType === 3) {
                var parent = $(this).parent();
                var html = parent.html();
                if(html) {
                    var newvalue = html.replace(re, function(){
                        var term = arguments[2],
                            before = arguments[1],
                            after = arguments[3];
                        term = '<abbr title="'+values[term]+'">'+term+'</abbr>';
                        return before + term + after;
                    });
                    parent.html(newvalue);
                }
         } else {
             $(this).contents().each(replaceNodeText);
         }
        }
        $("#content-core").contents().each(replaceNodeText);
    });
});
```

Lo primero que hacemos es comprobar si estamos en modo de edición, y si lo estamos, nos detendremos, ya que no queremos manipular el HTML que está siendo editado en TinyMCE o en cualquier campo de entrada.

Luego cargamos los términos del glosario con la siguiente llamada JSON: `/tutorial/@@rapido/glossary/records`

Usando el término valores que hemos cargado, construimos una expresión regular capaz de emparejar esos términos en cualquier texto.

Luego iteramos en los elementos principales del contenido de la página (`#content-core`), y cada vez que encontramos un nodo de texto, usamos nuestra expresión regular para reemplazar las palabras coincidentes con una etiqueta `<abbr>` donde el atributo `title` es la definición asociada.

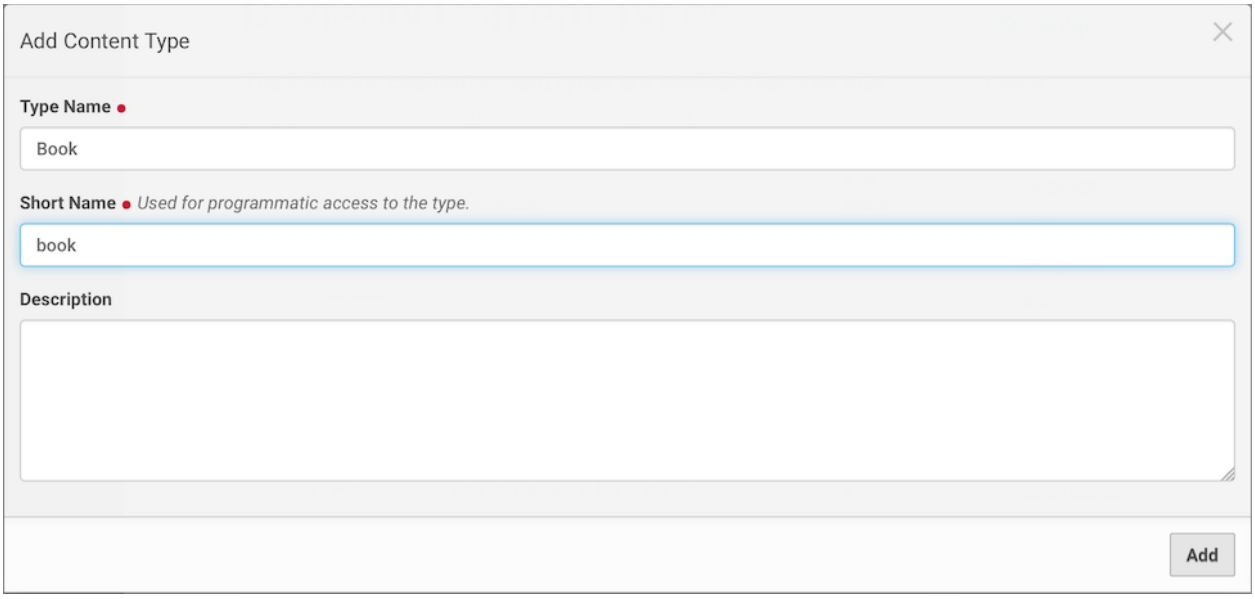
2.8.2 Utilice *Rapido* para crear un campo personalizado de *SearchableText*

Objetivo

Crear TTW (a través de la Web) a Tipo de contenido del *Book* donde los campos están indexados para una búsqueda de texto completo.

Tipo de contenido

Agregue un nuevo tipo de contenido *Book* en el panel de control *Tipos de contenido Dexterity* (http://localhost:8080/Plone/@@dexterity-types).



Edite la configuración de *Book* (http://localhost:8080/Plone/dexterity-types/book). En la pestaña *Comportamientos*, desmarque el comportamiento de los *Metadatos de Dublin Core* y *Guardar*. En la ficha *Campos*, haga clic en el botón *Editar modelo de campo XML* y reemplace el modelo XML por el siguiente:

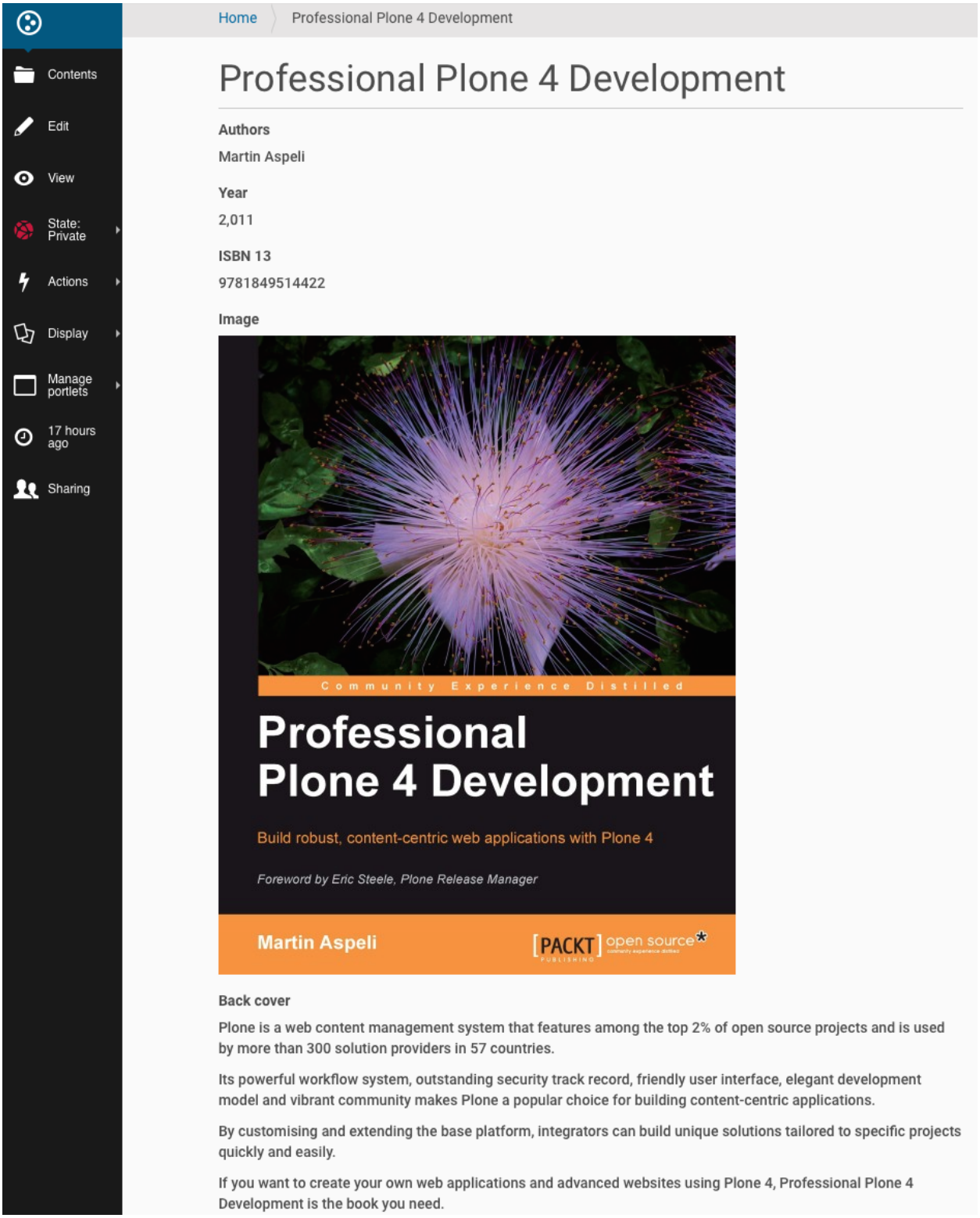
```
<model xmlns:form="http://namespaces.plone.org/supermodel/form"
       xmlns:i18n="http://xml.zope.org/namespaces/i18n"
       xmlns:lingua="http://namespaces.plone.org/supermodel/lingua"
       xmlns:marshal="http://namespaces.plone.org/supermodel/marshal"
       xmlns:security="http://namespaces.plone.org/supermodel/security"
       xmlns:users="http://namespaces.plone.org/supermodel/users"
```

(continues on next page)

(proviene de la página anterior)

```
      xmlns="http://namespaces.plone.org/supermodel/schema">
  <schema>
    <field name="title" type="zope.schema.TextLine">
      <description/>
      <title>Title</title>
    </field>
    <field name="authors" type="zope.schema.Text">
      <description/>
      <required>False</required>
      <title>Authors</title>
    </field>
    <field name="year" type="zope.schema.Int">
      <description/>
      <required>False</required>
      <title>Year</title>
    </field>
    <field name="isbn_13" type="zope.schema.TextLine">
      <description/>
      <max_length>13</max_length>
      <min_length>13</min_length>
      <required>False</required>
      <title>ISBN 13</title>
    </field>
    <field name="image" type="plone.namedfile.field.NamedBlobImage">
      <description/>
      <required>False</required>
      <title>Image</title>
    </field>
    <field name="back_cover" type="plone.app.textfield.RichText">
      <description/>
      <required>False</required>
      <title>Back cover</title>
    </field>
  </schema>
</model>
```

Ahora puede agregar contenido de *Book* en su sitio web (http://localhost:8080/Plone/++add++book).



Campo de búsqueda de texto completo

Si tiene un montón de libros en su sitio, le gustaría buscar en el nombre del autor o el contenido de la portada. Para ello, debemos proveer un método o campo `SearchableText` el cual le de el contenido del índice de texto completo.

Nosotros usaremos un *bloque rapido* y una *acción de regla de contenido disponible en el paquete rapido* para calcular este campo `SearchableText`.

Bloque Rapido

Vaya al panel de control Tema (http://localhost:8080/Plone/@@theming-controlpanel). Crea un nuevo tema llamado *MyTheme* con esta estructura siguiente.

```
index.html
manifest.cfg
rapido/
    book/
        blocks/
            fields.py
rules.xml
```

Busque, por ejemplo, en la sección Inheriting a new theme from Barceloneta de la documentación de Plone para obtener contenido de archivos `index.html`, `manifest.cfg` y `rules.xml`.

Contenido del archivo `fields.py`:

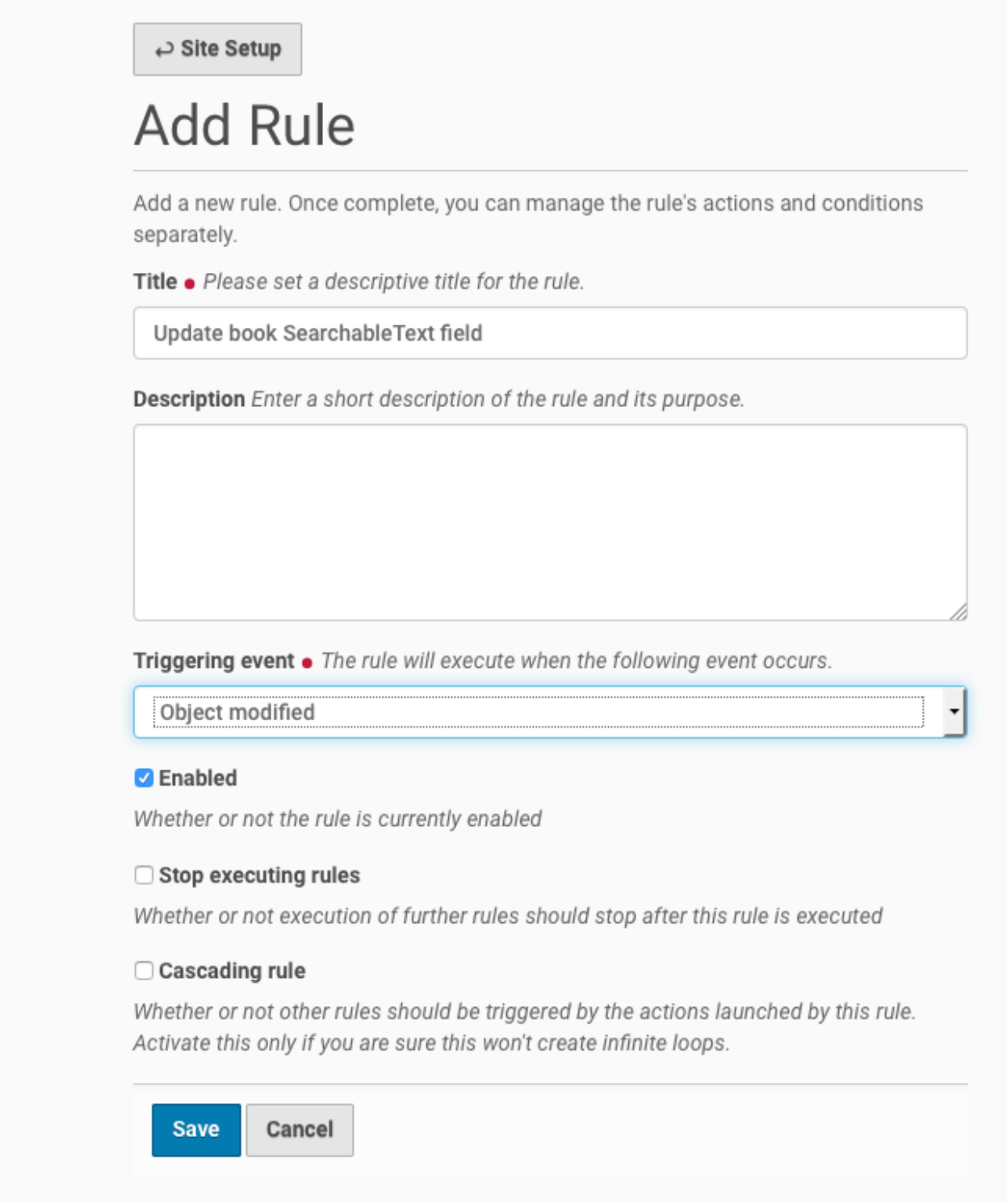
```
def update_searchabletext_field(context):
    transforms = context.api.portal.get_tool(name='portal_transforms')
    book = context.content
    back_cover_html = book.back_cover.output if book.back_cover else ""
    back_cover_plain = transforms.convertTo(
        'text/plain', back_cover_html, mimetype='text/html').getData()
    book.SearchableText = " ".join([
        book.title if book.title else "",
        book.authors if book.authors else "",
        str(book.year) if book.year else "",
        book.isbn_13 if book.isbn_13 else "",
        back_cover_plain
    ])
    book.reindexObject(idxs=['SearchableText'])
```

Usaremos la herramienta `portal_transforms` para convertir el campo HTML `back_cover` en texto plano. También necesitamos reindexar el contenido.

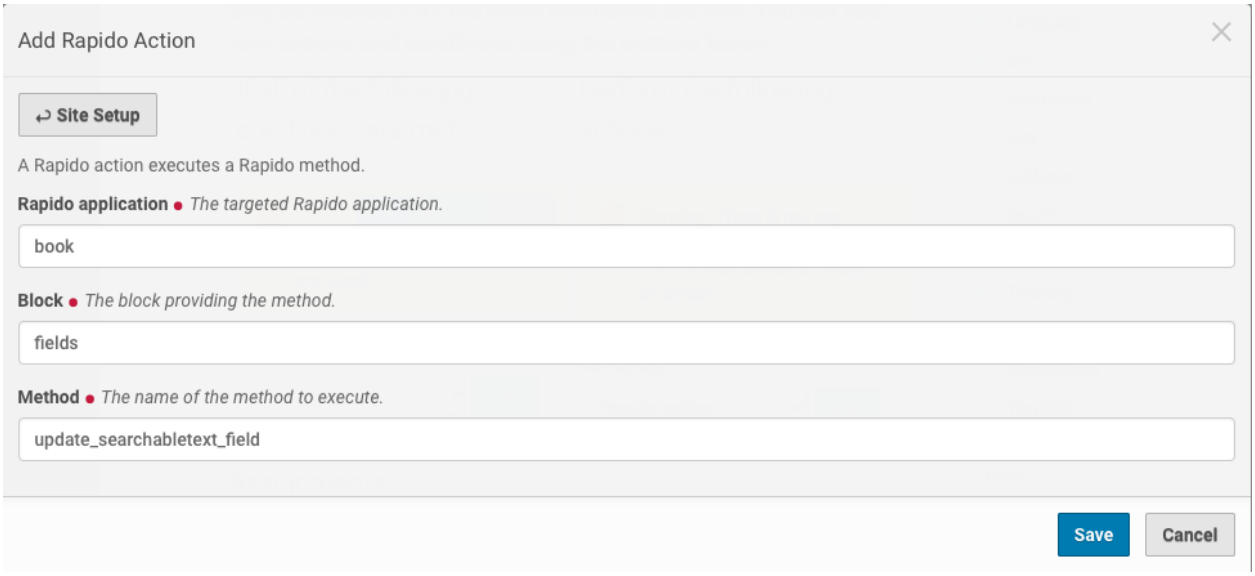
Acción de regla de contenido Rapido

Lo último que necesitamos es una *acción de regla de contenido rapido* que se usa en cada modificación del libro.

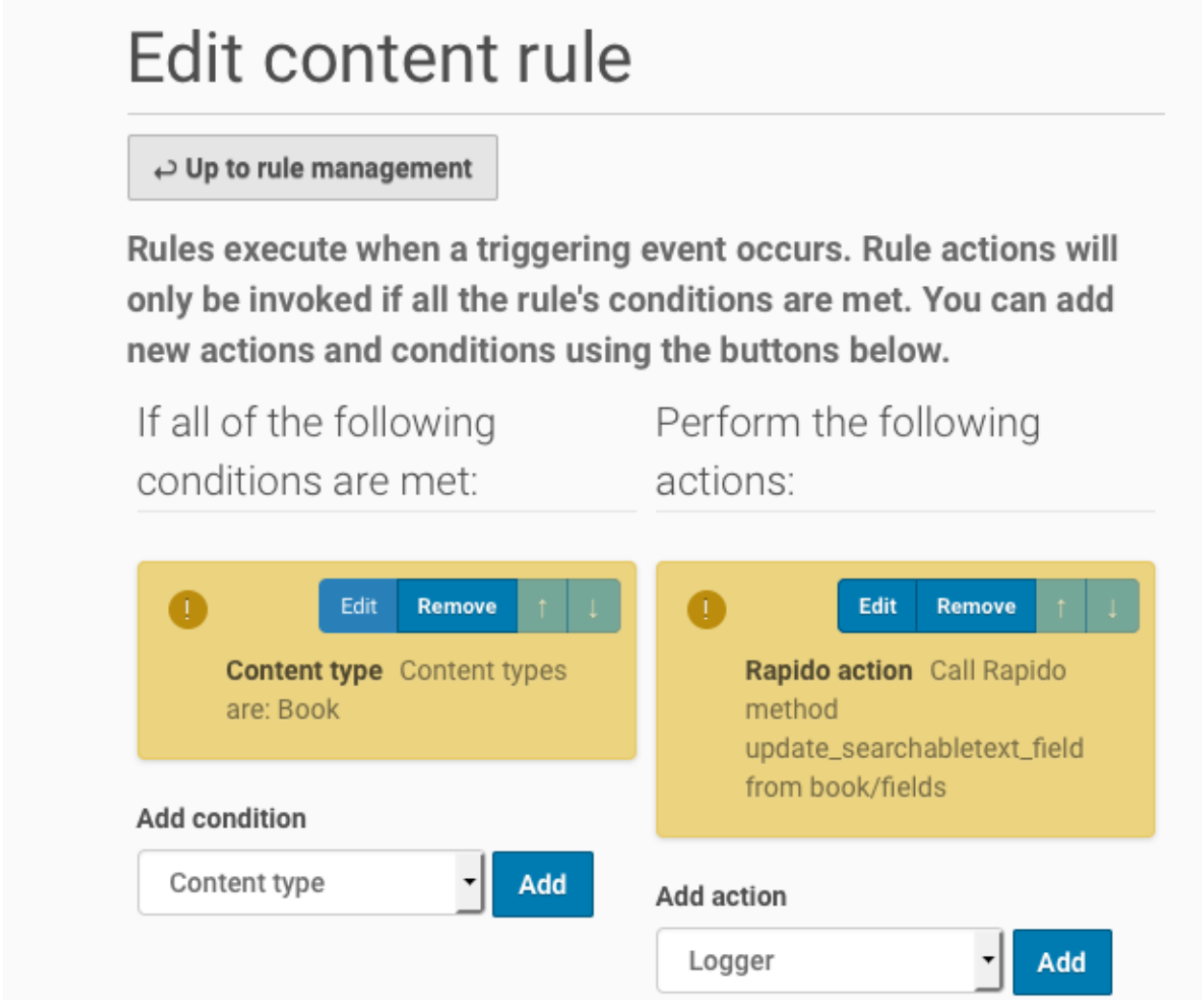
Vaya a las *Reglas de contenido* (http://localhost:8080/Plone/@@rules-controlpanel) y agregue una regla que se activa en el evento *Objeto modificado*.



Agregue una condición de *Tipo de contenido* en *Book*. Añade una *acción de Rapido*.



Asigne la regla de contenido en todo el sitio y *Guardar*.


Edit content rule
Up to rule management
Rules execute when a triggering event occurs. Rule actions will only be invoked if all the rule's conditions are met. You can add new actions and conditions using the buttons below.
If all of the following conditions are met:
Perform the following actions:
Edit
Remove
Content type Content types are: Book
Edit
Remove
Rapido action Call Rapido method update_searchabletext_field from book/fields
Add condition
Content type
Add
Add action
Logger
Add
Assignments
Info This rule is assigned to the following locations: Site
Configure rule
Title
Please set a descriptive title for the rule.
Update book SearchableText field
Description
Enter a short description of the rule and its purpose.

Ejercicio

Modifique el código anterior para calcular un campo *Descripción* que se utilizará en los listados de Plone

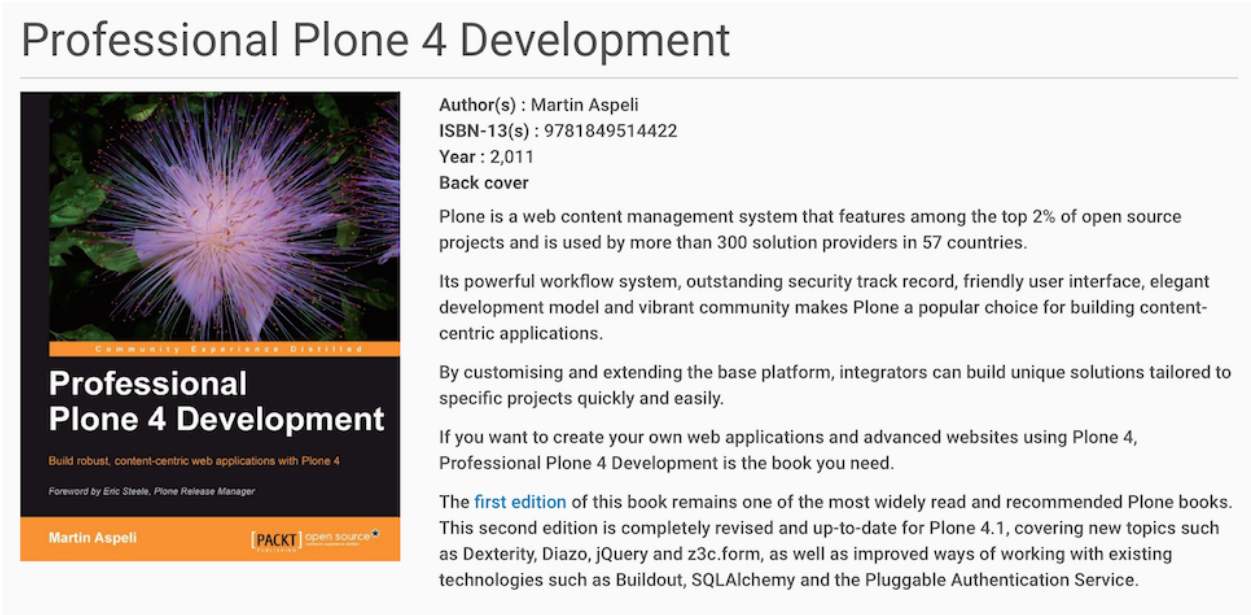
Vista personalizada de Book

Para crear una vista del tipo de contenido book personalizada, la solución más simple es utilizar una regla de *Diazo*.

Por ejemplo, puede agregar en el archivo `rules.xml` de su tema la siguiente `regla diazo`:

```
<rules css:if-content="body.template-view.portaltype-book">
  <replace css:content="#content-core" method="raw">
    <xsl:variable name="image_url">
      <xsl:value-of select="substring-before(//span[@id='form-widgets-image']/img/
↪@src,'view')"/>
    </xsl:variable>
    <div class="row">
        <div class="col-xs-12 col-sm-4">
            <xsl:if test="$image_url">
                <img src="{$image_url}@@images/image/large" style="width: 100%;" />
            </xsl:if>
        </div>
        <div class="col-xs-12 col-sm-8">
            <div><strong>Author(s) : </strong><xsl:copy-of css:select="#form-widgets-
↪authors" /></div>
            <div><strong>ISBN-13(s) : </strong><xsl:copy-of css:select="#form-widgets-
↪isbn_13" /></div>
            <div><strong>Year : </strong><xsl:copy-of css:select="#form-widgets-year"␣
↪/></div>
            <div><xsl:copy-of css:select="#formfield-form-widgets-back_cover" /></div>
        </div>
    </div>
  </replace>
</rules>
```

Nuestras nuevas vistas de libros personalizadas:



2.9 Licencia