Cookbook - CMS Basics

Pages

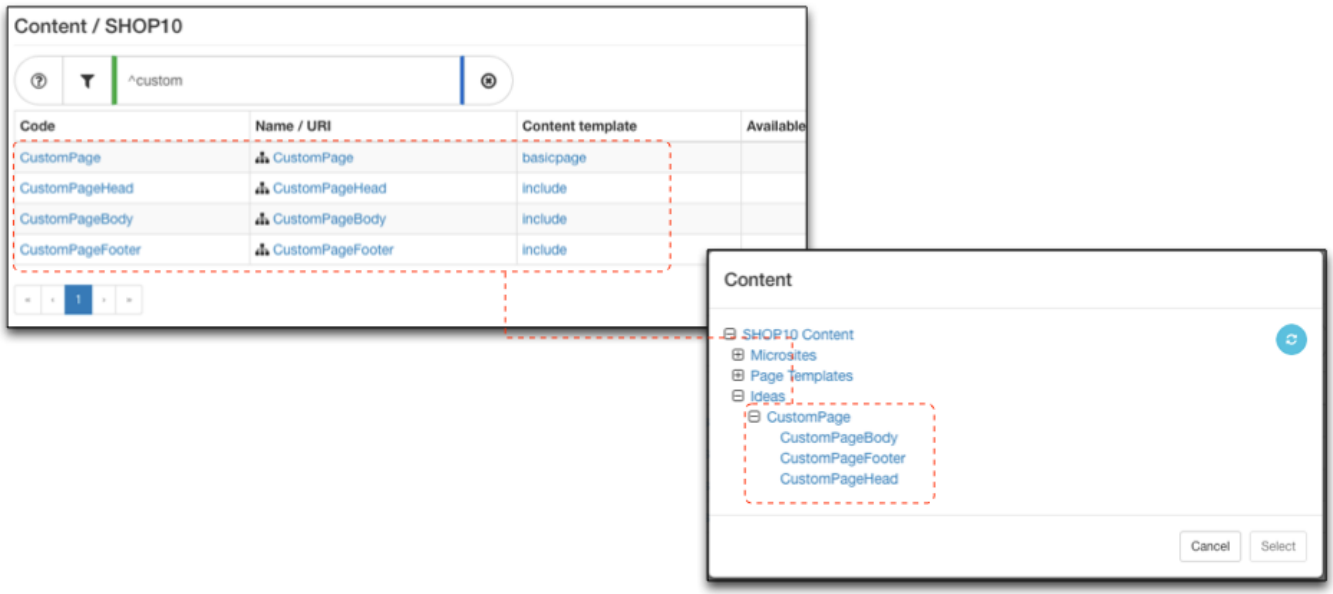
Every new content object created in CMS defaults to **content** template, which makes it available publicly as a content page (provided availability configurations allow so).

Default theme's content template is a page with header, body section and footer. The content body from content object is processed and inserted into this template thus populating the middle section. This behaviour is the behaviour of default theme.

Concepts and ideas

Suppose we do not want to use default behaviour with content page having a fixed layout with header, body and footer. Custom themes can override this behaviour to use different templates and provide alternative ways of rendering content body in the final page (e.g. combining several content objects together to form a single page).

For example custom theme can define a template **basicpage**, which does not use the content object but sees it as a group (i.e. this content object must have 3 children: head, body and footer) and then compose a page just from these content elements.



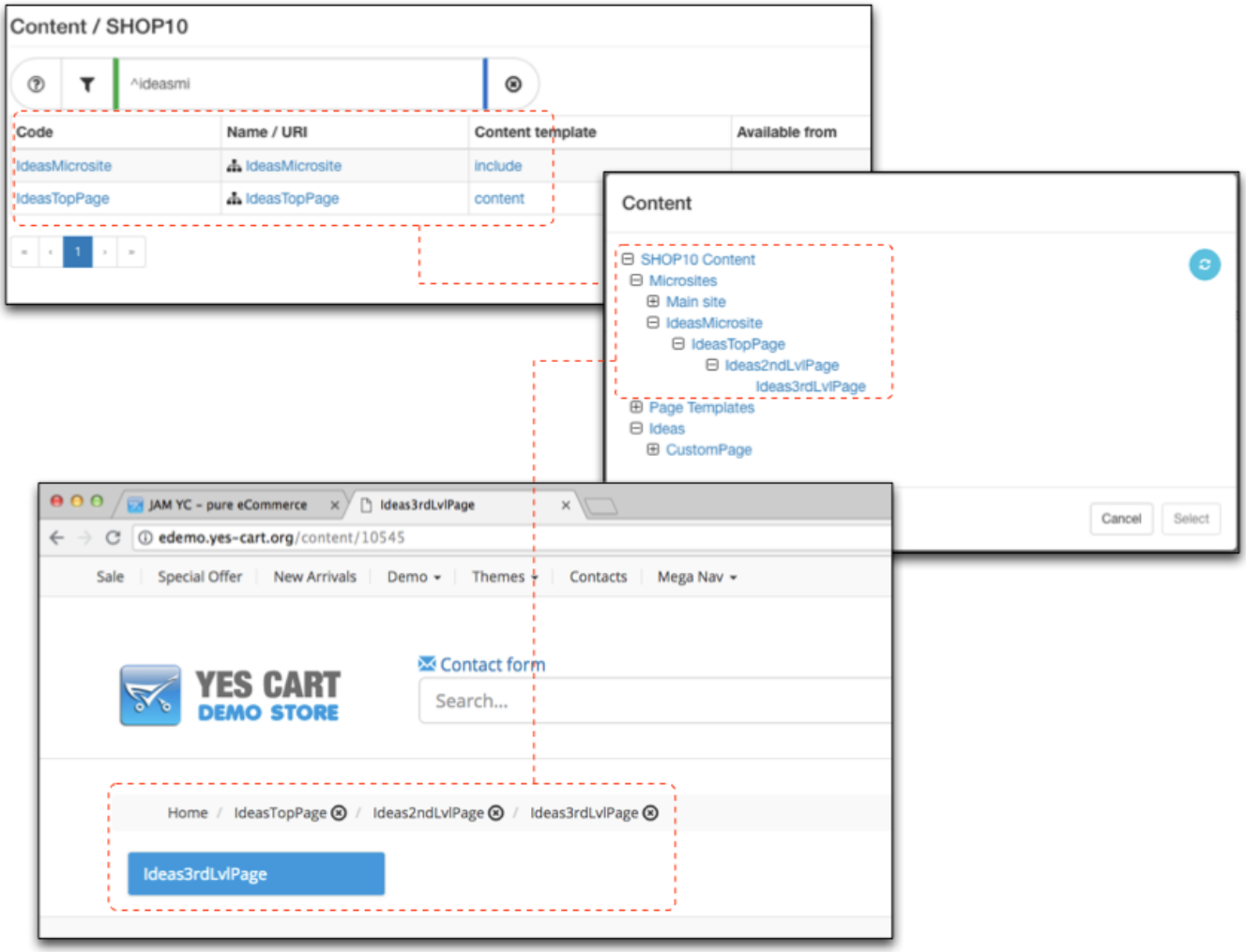
Note that in the configurations we use **basicpage** template for the CustomPage content, which has three sub content which are configured as **inc lude** (to prevent direct public access). The logic for handling **basicpage** would be custom implementation in custom theme. In principle it will be a blank page with just essential CSS and JavaScript elements included in the template and then the **<body>** of the page would simply have three sections that will contain the bodies of CustomPageHead, CustomPageBody and CustomPageFooter

Microsites and page hierarchies

All content is derived from the root content object. For example in demo shop **SHOP10 Content** is the root content for all content of this shop. This means that inevitably every content we create is a child of the root content and thus in terms of breadcrumbs all content points to this top level object.

In terms of business we would like to have individual hierarchies for specific site sections such as: about us section of the site that could contain multiple pages with store finder, contact details and maybe feedback form, or for a crafting company there could be section with workshops and ideas, or maybe business wants to publish a section with news and updates (or a blog).

For this purpose we can build separate branches of these microsites and then cut-off the breadcrumbs by making parent content of these microsites use **include** template.



In the example it can be seem that IdeasTopPage is under SHOP10Content > Mictosites > IdeasMicrosite, however because IdeasMicrosite has template configuration set to **include** the breadcrumb is cut-off, so that on the web site the path to Ideas3rdLvlPage is Home > IdeasTopPage > Ideas2ndLvlPage > Ideas3rdLvlPage

Includes

Includes refers to content objects that use **include** template configured and are references from a theme template as a part of the page structure. Consider the following example of including top navigation in storefront wicket (SFW) application.

Template defines **headerNav** container

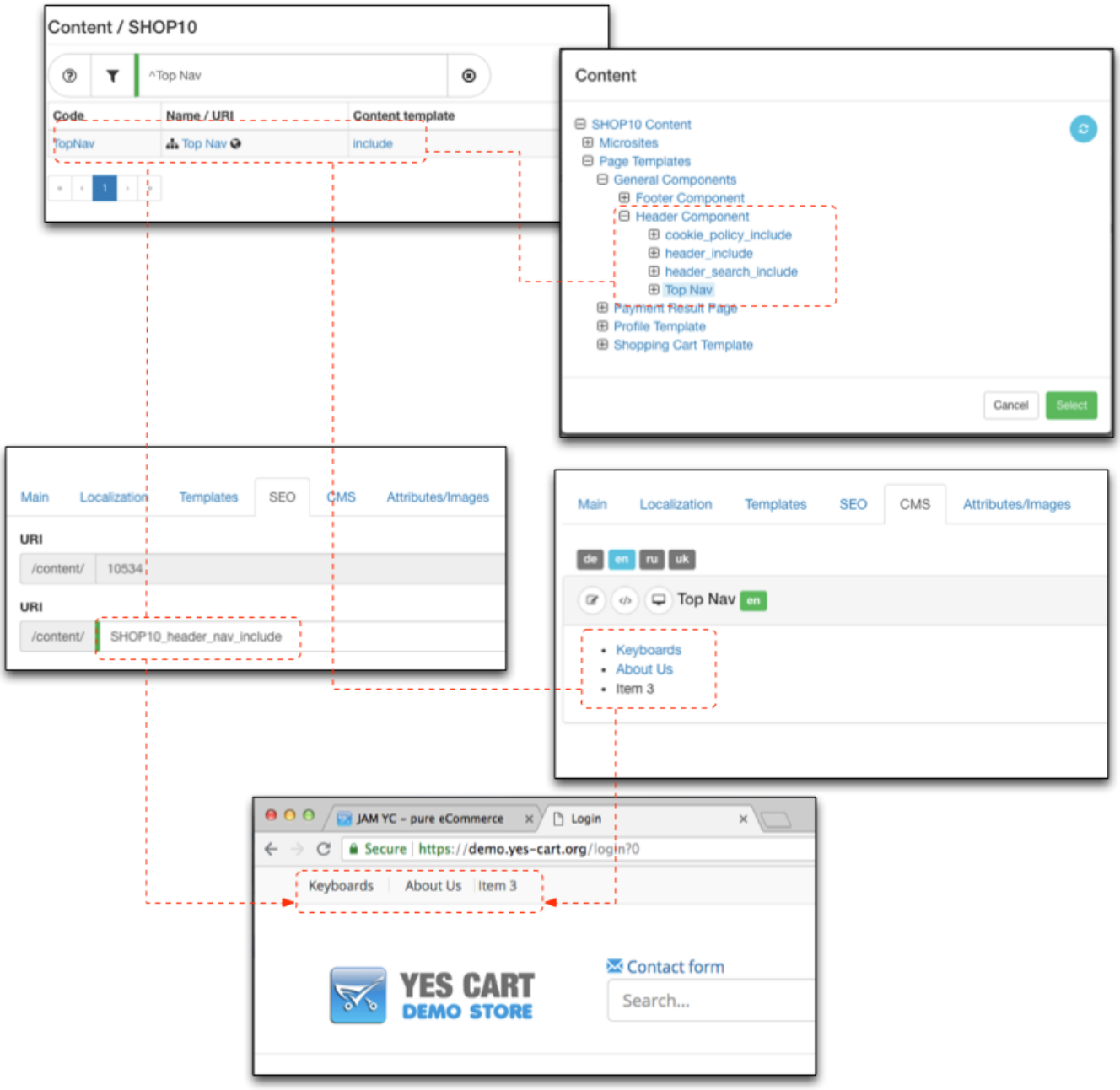
```
    <nav class="top-menu">
         <div class="container no-padding">
             <div class="col-xs-12 col-sm-12 col-md-8 col-lg-8 no-margin">
                 <wicket:container wicket:id="headerNav">
                     [header nav]
                 </wicket:container>
             </div>
```

StandardHeader component populates this value with content body of **header_nav_include**.

```
       String content = contentServiceFacade.getContentBody(
                "header_nav_include", getCurrentShopId(),
getLocale().getLanguage());

        return new Label("headerNav", content).setEscapeModelStrings(false);
```

Everything else is configured in the CMS. We set shop specific URI for the include to: **SHOP10_ header_nav_include** and enter the navigation menu content into the body, which result in that area of the page to include this element.



Note that it does not matter where content include is in the CMS hierarchy as it is resolved solely by URI. However it is best practice to group the components logically so that they are easier to manage.

Role of URI

There is no distinction between URI for content pages or content includes, however it is a requirement to have unique URI for all content as SEO URLs are resolved in a generic way. Therefore there is a limitation of what URI you can use for the content.

Due to this limitation content includes have to have shop specific URIs, which means all documented in theme templates include URIs have to be prefixed with shop code followed by underscore.

In top menu include example which was using **header_nav_include** in code we used **SHOP10_ header_nav_include** instead. Theme can be

shared by many shops, however each shop must have its own content with own URI, thus prefixing it allows to have unique URI per shop and allow content elements to be defined for each individual shop.

Configuring include on SFW

Say we want to create CMS include with "my_cms_include" that will be included in our custom component.

Define container element in Wicket HTML template, e.g. **myCmsContainer**

```
<wicket:container wicket:id="myCmsContainer">[my cms
container]</wicket:container>
```

Add container element as Label containing body of the CMS content object, retrieved by content service using URI, e.g. "my_cms_include"

```
String content = contentServiceFacade.getContentBody(
    "my_cms_include", getCurrentShopId(), getLocale().getLanguage());

addOrReplace(new Label("myCmsContainer",
content).setEscapeModelStrings(false));
```

If we want dynamic content to be included the java code should be updated to use dynamic body and supply all parameters to be used in this dynamic content.

```
final Map<String, Object> contentParams = new HashMap<>();
...
// populate contentParams map
...
String content = contentServiceFacade.getDynamicContentBody(
    "my_cms_include", getCurrentShopId(), getLocale().getLanguage(),
contentParams);

addOrReplace(new Label("myCmsContainer",
content).setEscapeModelStrings(false));
```

Configure CMS object with URI [shop code]_my_cms_include .

Configuring include on SFG SaaS

Say we want to create CMS include with "my_cms_include" that will be included in our custom component.

Use ctx.content('my_cms_include') for plain HTML or ctx.dynamicContent('my_cms_include', ['p1':'value1', 'p2object': myObject])

For example template which uses include would look like so:

```
 div('class': 'container') {
      mkp.yieldUnescaped(ctx.content('my_cms_include'));
}
```

```
 div('class': 'container') {
      mkp.yieldUnescaped(ctx.dynamicContent('my_cms_include',
['p1':'value1', 'p2object': myObject]));
}
```

SFG implementation does not require any java changes and can just include content either plain or dynamic right in the template, which results in more flexible and faster implementation of storefront changes and theme development in general.