

Extending the Demo

Creating a new component with custom fields [↗](#)

This guide will walk you through creating a new component with custom fields

Composable Storefront [↗](#)

First you need to create the component in Composable Storefront.

Step 1: Define the Component Model [↗](#)

Create a model for the new component:

```
src/app/model/cms.model.ts

1 import { CmsComponent } from '@spartacus/core';
2 import { CmsBannerComponentMedia } from '@spartacus/core/src/model/cms.model';
3 export interface CmsTextImageComponent extends CmsComponent {
4   text?: string;
5   image?: CmsBannerComponentMedia;
6 }
```

Step 2: Create the Component [↗](#)

1. Create the Component File [↗](#)

```
src/app/cms-components/content/text-image/text-image.component.ts

1 import { ChangeDetectionStrategy, Component } from '@angular/core';
2 import { CmsComponentData } from '@spartacus/storefront';
3 import { CmsTextImageComponent } from '../../../model/cms.model';
4 @Component({
5   selector: 'cx-text-image',
6   templateUrl: './text-image.component.html',
7   styleUrls: ['./text-image.component.scss'],
8   changeDetection: ChangeDetectionStrategy.OnPush,
9 })
10 export class TextImageComponent {
11   constructor(public component: CmsComponentData<CmsTextImageComponent>) {}
12 }
```

2. Create the Template File [↗](#)

```
src/app/cms-components/content/text-image/text-image.component.html

1 <div *ngIf="component.data$ | async as data" class="container">
2   <h2>{{data.text}}</h2>
3   
4 </div>
```

3. Create the Stylesheet [↗](#)

```
src/app/cms-components/content/text-image/text-image.component.scss

1 .container {
2   display: flex;
```


```

3   flex-direction: column;
4   justify-content: center;
5   align-items: center;
6 }

```

4. Create a normalizer for the image [↗](#)

Since Contentful structures images differently, we need a normalizer to transform the image data into the format used in Angular.

 src/app/cms-components/content/text-image/text-image-component-normalizer.ts


```

1  import { Injectable } from '@angular/core';
2
3  import { CmsComponent } from '@spartacus/core';
4  import { CmsBannerComponentMedia } from '@spartacus/core/src/model/cms.model';
5
6  import { Asset, Entry } from 'contentful';
7
8  import { ComponentSkeleton } from '../../../contentful/core/content-types';
9  import { isAsset } from '../../../contentful/core/type-guards';
10 import { CmsTextImageComponent } from '../../../model/cms.model';
11
12 @Injectable({ providedIn: 'root' })
13 export class TextImageComponentNormalizer {
14   convert(source: Entry<ComponentSkeleton, undefined, string>, target: CmsTextImageComponent): CmsComponent {
15     if (source.sys.contentType.sys.id === 'CmsTextImageComponent' && isAsset(source.fields?.['image'])) {
16       target.image = this.normalizeMediaAsset(source.fields['image']);
17     }
18     return target;
19   }
20
21   private normalizeMediaAsset(media: Asset<undefined, string>): CmsBannerComponentMedia | undefined {
22     return {
23       altText: '',
24       code: '',
25       mime: media.fields.file?.contentType ?? '',
26       url: media.fields.file?.url ?? '',
27     };
28   }
29 }

```

Step 3: Register the Component and inject the normalizer [↗](#)

Create a module to register the new component and provide the normalizer in Spartacus.

 src/app/cms-components/content/text-image/text-image.module.ts

```

1  import { CommonModule, NgOptimizedImage } from '@angular/common';
2  import { NgModule } from '@angular/core';
3  import { CMS_COMPONENT_NORMALIZER, CmsConfig, provideDefaultConfig } from '@spartacus/core';
4  import { TextImageComponent } from './text-image.component';
5  import { TextImageComponentNormalizer } from './text-image-component-normalizer';
6
7  @NgModule({
8    imports: [CommonModule, NgOptimizedImage],
9    providers: [
10     provideDefaultConfig(<CmsConfig>{
11       cmsComponents: {
12         CmsTextImageComponent: {

```


```

13     component: TextImageComponent,
14   },
15 },
16 }},
17 {
18   provide: CMS_COMPONENT_NORMALIZER,
19   useExisting: TextImageComponentNormalizer,
20   multi: true,
21 },
22 ],
23 declarations: [TextImageComponent],
24 exports: [TextImageComponent],
25 })
26 export class CmsTextImageModule {}

```

Step 4: Import the Module into the Spartacus Feature Module [↗](#)

Finally, add the new component module to the **Spartacus Feature Module**:


 src/app/spartacus/spartacus-features.module.ts

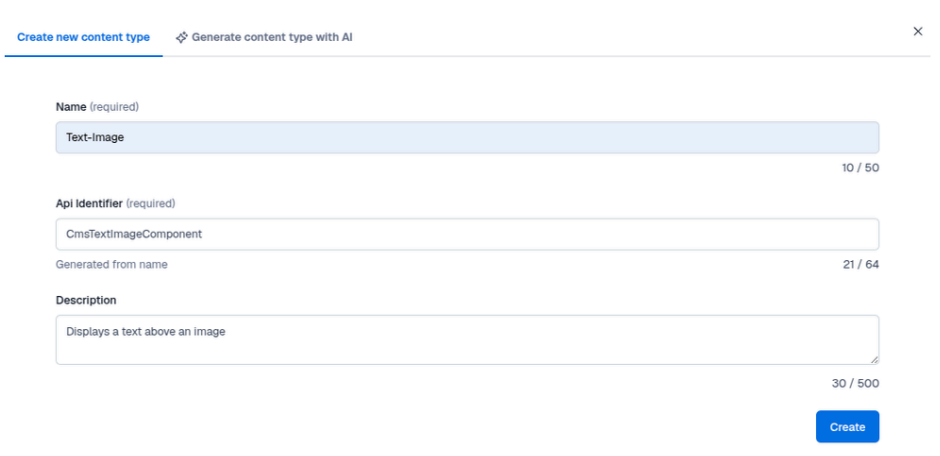
[🔗](#) More details: [SAP Help - Creating a New Component](#)

Contentful [↗](#)

Now that we've created the component in **Composable Storefront**, the next step is to define the component template and content in **Contentful**.

Step 1: Create the Component Template [↗](#)

1. Open **Contentful**, go to **Content Models**, and click **Create Content Type**.
2. Fill in the required details:
 - **Name** ("Text-Image")
 - **API Identifier** ("CmsTextImageComponent").
 - **Description**
 -  The **API Identifier** must match the name ("CmsTextImageComponent") used when registering the component module in **Composable Storefront**.



3. Add two fields:
 - **Text**
 - Type: "Text" => "Short text, exact search"
 - Name: "text"
 - Field ID "text"

Add field Short text

Name (required)

Field ID

text

text

Appears in the entry editor

4 / 50

Appears in API responses

4 / 64

These settings can't be changed later.

Type

☒ Short text, exact search

☐ Long text, full-text search

☐ List

Enables sorting, 256 characters max.
Use this for titles, names, tags, URLs, e-mail addresses

No sorting, 50k characters max.
Use this for descriptions, text paragraphs, articles

Select this if there is more than one value to store, like several names or a list of ingredients.
API response will include a separate block for each field.

Change field type

Add and configure

◦ Image

- Type: "Media" => "One file"
- Name: "image"
- Field ID "image"

Add field Media

Name (required)

Field ID

image

image

Appears in the entry editor

5 / 50

Appears in API responses

5 / 64

These settings can't be changed later.

Type

☒ One file

☐ Many files

For example, a single photo or one PDF file.

For example several photos, PDF files, etc.
API response will include a separate block for each field.

Change field type

Add and configure

4. Click **Create** to save the new template.

Step 2: Create the New Component [↗](#)

Now, create a new component based on the template.

Editor
References
Taxonomy
New
Tags

text | English

Demo Image

10 characters

Maximum 256 characters

media

Published

POWER DRILLS

1. Create a new **Text-Image** content entry.

2. Add **text** and **image** values to be displayed in the component.
3. Click **Publish** to save and activate the component, including the image.

Creating a New Page with a Custom Template

This guide will walk you through setting up a new page using a custom layout with new content slots.

Composable Storefront

To add a new page with a custom template in Composable Storefront, you need to configure both the **layout** and the **routing**. Once set up, you'll provide these configurations to the Composable Storefront application.

Step 1: Configure the Layout

Create a custom layout configuration file:


 src/app/config/layout-config.ts

```
1 import { LayoutConfig } from '@spartacus/storefront';
2 export const layoutConfig: LayoutConfig = {
3   layoutSlots: {
4     DemoPageTemplate: {
5       slots: ['MainContent'],
6     },
7   },
8 };
```

 More details: [SAP Help - Page Layout](#)

Step 2: Configure Routing

Create a custom routing configuration file:

 src/app/config/routing-config.ts

```
1 import { RoutingConfig } from '@spartacus/core';
2 export const routingConfig: RoutingConfig = {
3   routing: {
4     routes: {
5       demo: {
6         paths: ['demo'],
7       },
8     },
9   },
10 };
```

 More details: [SAP Help - Routing](#)

Step 3: Provide the Config to Composable Storefront

Create a new configuration module:

 src/app/config/config.module.ts

```
1 import { NgModule } from '@angular/core';
2 import { provideConfig } from '@spartacus/core';
3 import { layoutConfig } from './layout-config';
4 import { routingConfig } from './routing-config';
5 @NgModule({
6   providers: [provideConfig(layoutConfig), provideConfig(routingConfig)],
7 })
```

```
8 export class DemoConfigModule {}
```

Finally, **import the** `DemoConfigModule` **in your** `AppModule` to apply the configurations.

Contentful [↗](#)

Now that we've configured the layout and routing, the next step is to define the page template and content slots in **Contentful**.

Step 1: Add the Page Template [↗](#)

1. Open **Contentful** and go to your **Content Models**.
2. Edit the **CMSPage** model.
3. Add the new template:
 - Find the **Template** field and edit it.
 - Under **Accept only specified values**, enter the name of the new template.

☒ **Accept only specified values**

You won't be able to publish an entry if the field value is not in the list of specified values

Hit enter to add a value

AccountPageTemplate X

CartPageTemplate X

CategoryPageTemplate X

CheckoutLoginPageTemplate X

ContentPage1Template X

ErrorPageTemplate X

LandingPage2Template X

LoginPageTemplate X

MultiStepCheckoutSummaryPageTemplate X

MyAccountViewPageTemplate X

OrderConfirmationPageTemplate X

ProductDetailsPageTemplate X

ProductGridPageTemplate X

ProductListPageTemplate X

SearchResultsGridPageTemplate X

SearchResultsListPageTemplate X

DemoPageTemplate X

- Click **Confirm** to save your changes.

Step 2: Add New Content Slots [↗](#)

1. In the **CMSPage** model, create a new field of type **Reference**
 - a. **Name:** "MainContent"
 - b. **Field ID:** "MainContent" (⚠️ The **Field ID** must match the slot name from the layout configuration)
2. Choose **Many references**, allowing multiple components to be added to the slot.

Add field Reference X

Name (required)

MainContent

Field ID

MainContent

Appears in the entry editor

11 / 50

Appears in API responses

11 / 64

⚠️ These settings cannot be changed later.

Type

☐ One reference

For example, a blog post can reference only one author.

☒ Many references

For example, a blog post can reference several authors. The API response will include a separate block for each field.

Change field type

Add and configure

Step 3: Add and Configure the New Slot [↗](#)

Use the **FlexFields app** from the **Contentful Marketplace** to manage slot selection.

The screenshot shows the 'FlexFields App Config' interface. At the top, there's a header with the app logo and name. Below it, a button says '+ Add new rule(s) below'. The configuration is divided into sections: 'For content type' (set to 'CMS Page'), 'With field' (set to 'Template'), 'Condition' (set to 'is equal'), and 'DemoPageTemplate'. Under 'Hide field:', there's 'Within content type' (set to 'CMS Page (Same Entry)') and 'Hide field' (set to 'Select one or more Items'). A dropdown menu is open for 'Hide field', showing a list of content slots: 'BodyContent', 'SideContent', 'LeftContentSlot', 'RightContentSlot', and 'MainContent'. The first four are checked, and 'MainContent' is unchecked. Below the configuration, there's a section for 'Current rules (18 rules)' with a preview of a rule: 'If content type "CMS Page" has Template which is empty then in entity SearchResultsListSlot, SearchResultsGridSlot, Summary, UpSelling, C RightContentSlot" field(s)'. To the right of the rule preview, there's a list of content slots: 'Section2, Section2/...', 'opContent, Middle...', and 'MainContent'.

1. Navigate to **Apps → Installed** and configure the app.
2. Add a new rule for the custom template:
 - Hide all content slots **except** the ones that should be part of the template.
3. Edit all other rules to **exclude** the newly created slots.

Step 4: Create the Content Page [↗](#)

Now that the new page template is ready, we can create a content page based on it.

FlexFields
Editor
References
Tags
Taxonomy

Internal Name (required)

DemoPage

8 characters
Maximum 256 characters

Title | English (required)

Demo Page

9 characters
Maximum 256 characters

Title | German (required)

Demo Seite

10 characters
Maximum 256 characters

Slug | English (required)

demo

Slug | German (required)

demo

Preview URL (required)

demo

4 characters
Maximum 256 characters

Template (required)

DemoPageTemplate

MainContent

Text-Image

Published

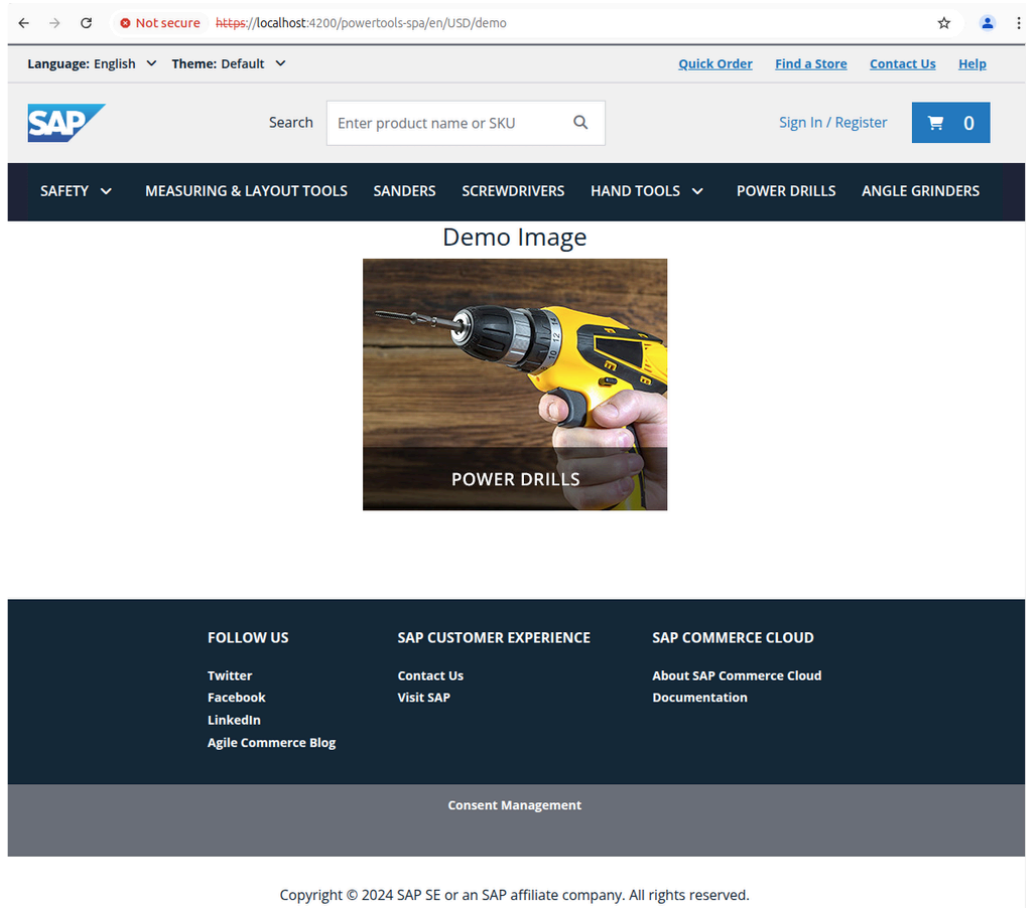
...

Demo Image

+ Add content

1. Create a new **CMSPage** content entry.
2. Select the **Template** from the dropdown. This will display the available slots.
3. Fill in the required fields:
 - **Internal Name:** A name to identify the component.
 - **Title:** The headline displayed in the browser tab.
 - **Slug:** Must match the route set in the routing configuration, which in this example is “**demo**”
 - **Preview URL:**
 - **For static pages:** Use a simple **slug** that matches the route.
 - Example: “demo” → Accessible at `https://localhost:4200/demo`
 - **For dynamic pages:** Use a **specific example** of the dynamic route.
 - Example: “product/3755219/PSR%20960” → Previews a **product detail page** for a specific product
4. Add the previously created **Text-Image** component to the **MainContent** content slot.
5. **Publish the page.**

Step 5: Validate the Setup [↗](#)



1. Start the development server.
2. Navigate to your new route:
 - <https://localhost:4200/powertools-spa/en/USD/demo>
3. Confirm that the components appear correctly on the page.