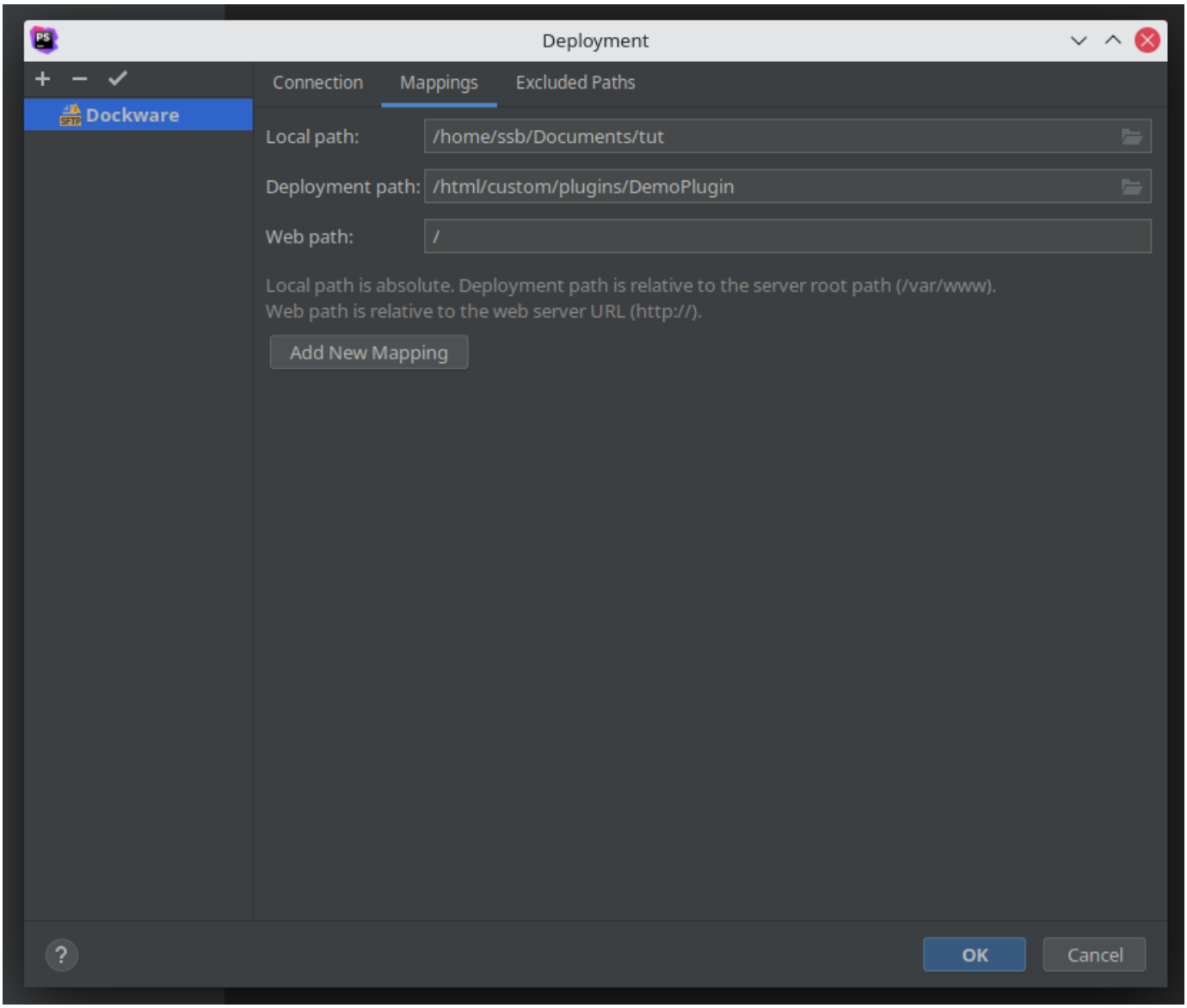
SW6 Plugin Entwicklung

1. Projekt Setup

Wir erstellen ein neues Projekt in PhpStorm, fügen die Vendor Dateien hinzu und richten das Deployment für Dockware ein.



2. Plugin Basis

2.1 Die composer.json Datei

Die composer.json Datei liegt im Root Verzeichnis unseres Plugins und beinhaltet die Grundinformationen die Shopware benötigt und das Plugin zu laden.

Wichtig ist hier der “autoload” Eintrag. Dieser entscheidet wie der Code geladen wird (im Normalfall ist dies unser Vendor Prefix und der Namespace des Plugins).

Unter “extra” > “shopware-plugin-class” wird die Hauptklasse des Plugins festgelegt.

```
{
  "name": "bow/tut-plugin",
  "description": "BOW Tutorial Plugin",
  "version": "v0.1.0",
  "type": "shopware-platform-plugin",
  "license": "proprietary",
  "authors": [
    {
      "name": "BOW - Bayerisches Onlinewerk",
      "homepage": "https://bow-agentur.de",
      "role": "Manufacturer"
    }
  ],
  "require": {
    "shopware/core": "6.3.*"
  },
  "autoload": {
    "psr-4": {
      "Bow\\BowTutPlugin\\": "src/"
    }
  },
  "extra": {
    "shopware-plugin-class":
"Bow\\BowTutPlugin\\BowTutPlugin",
    "manufacturerLink": {
      "de-DE": "https://bow-agentur.de",
      "en-GB": "https://bow-agentur.de"
    },
    "label": {
      "de-DE": "BOW Tutorial Plugin",
      "en-GB": "BOW Tutorial Plugin"
    }
  }
}
```

2.2 Plugin Basis Klasse

Die Basis Klasse des Plugins liegt normalerweise direkt im src Verzeichnis

```php
<?php declare(strict_types=1);

namespace Bow\BowTutPlugin;

use Shopware\Core\Framework\Plugin;

class BowTutPlugin extends Plugin {
}
```

Und damit ist das Plugin deklariert und lässt sich installieren und aktivieren.

3. Templates

3.1 Templates überschreiben

Die Standard Templates für die Storefront sind hier zu finden und lassen sich überschreiben in dem im `Resources` Ordner eine Datei in der gleichen Ordnerstruktur angelegt wird. Das bedeutet das wir eine Datei, wie zum Beispiel die Footer Datei überschreiben können in dem wir eine TWIG Datei mit dem gleichen Namen und dem gleichen relativen Pfad anlegen, in diesem Fall unter `rc/Resources/views/storefront/layout/footer/footer.html.twig`.

Da wir aber meistens nur einen (oder mehrere) Block überschreiben wollen können wir die orginale Datei mit `{% sw_extends '@Storefront/PFAD' %}` erweitern. Wir ergänzen jetzt den Footer mit einem neuen `<div>` Element im Footer.

```twig
{% sw_extends '@Storefront/storefront/layout/footer/footer.html.twig' %}

{% block layout_footer_copyright %}
    {% parent() %}
    <div class="tut-footer">
        Footer Extension
    </div>
{% endblock %}
```

⚠ Nachdem neue Template Dateien hinzugefügt wurden muss die Cache gelöscht werden (nach dem Bearbeiten einer existierenden Datei ist das im Normalfall nicht nötig)

Wir verwenden die `block` Direktive um den zu überschriebenen Block auszuwählen, da wir den original Inhalt aber nicht verlieren möchten verwenden wir die Twig Funktion `parent()` um den Inhalt des davor gerenderten Templates zu importieren.

3.2 SCSS einfügen

Wir haben unserem `<div>` Element auch eine CSS Klasse verpasst, diese wollen wir natürlich jetzt auch nutzen. Hierzu legen wir eine SCSS Datei im Ordner `Resources/app/storefront/src/scss` an.

```
.tut-footer {
  background-color: rgba(0,0,0,.8);
  color: #eee;
  max-width: 720px;
  padding: 4px 8px;
  margin: 8px auto;
  transition: transform .5s ease-in-out;
  cursor: pointer;

  &.active {
    transform: rotateZ(180deg);
  }
}
```

Sämtliche SCSS Dateien in diesem Ordner werden automatisch geladen und in das Theme integriert. Um das Theme zu kompilieren müssen wir den Befehl `theme:compile` in der Shopware Console ausführen. Wir können uns mit dem Dockware Image entweder über einen Terminal verbinden oder den integrierten Terminal in PhpStorm verwenden (Tools > Start SSH Session…)

```
./html/bin/console theme:compile
```

3.3 JavaScript einfügen

Shopware lädt JavaScript für die Storefront aus der `src/Resources/app/storefront/src/main.js` Datei.

JS Dateien für das Admin Backend werden anders angelegt und in einem anderen Punkt angesprochen.

Nachdem wir die main.js Datei befüllt haben (zum Beispiel mit einem `console.log` Statement) müssen wir die Storefront neu kompilieren. Dazu verwenden wir den `build-storefront` Script:

```
./html/bin/build-storefront.sh
```

Shopware liefert ein Pluginsystem mit, dies ist sehr nützlich um JavaScript an bestimmte Elemente zu hängen, und somit die Last auf dem Browser zu verringern. Dazu legen wir ein Shopware JS Plugin an.

Wir erstellen eine neue Datei unter `src/Resources/app/storefront/src/footer-plugin/footer-plugin.plugin.js`:

```
import Plugin from 'src/plugin-system/plugin.class';

export default class FooterPlugin extends Plugin {
    init() {
        this.el.onclick = () => {
            this.el.classList.toggle('active');
        }
    }
}
```

Damit unser JS Plugin geladen wird muss dieses in der `main.js` registriert werden:

```
import FooterPlugin from './footer-plugin/footer-plugin.plugin';

console.log('main.js geladen!');

const PluginManager = window.PluginManager;
PluginManager.register('FooterPlugin', FooterPlugin, '.tut-footer');
```

3.4 Template Dateien einbinden

Neu definierte Templates können über `{% sw_include %}` eingebunden werden. Hierfür legen wir ein neues Template unter `src/Resources/views/storefront/layout/footer/tut-footer.html.twig` an:

```
{% block tut_footer %}
    <div class="tut-footer">
        {{ footerText }}
    </div>
{% endblock %}
```

Und wir ändern unsere `footer.html.twig` um unsere neue Datei zu importieren:

```
{% sw_extends '@Storefront/storefront/layout/footer/footer.html.twig' %}

{% block layout_footer_copyright %}
    {{ parent() }}
    {% sw_include '@Storefront/storefront/layout/footer/tut-footer.html.
twig' with {
        footerText: 'Footer Extension'
    } only %}
{% endblock %}
```

Im `sw_include` fügen wir eine Variable hinzu (`footerText`), durch den `only` Parameter wird festgelegt das nur diese Variable übergeben soll, ohne diesen Parameter sind im Child Komponenten auch sämtliche Variablen des Parents verfügbar.

4. CMS Elemente

CMS Elemente bestehen aus zwei Hauptkomponenten:

- Adminbereich Vue Komponent
- Storefront Twig Komponent

4.1 Adminbereich Komponent

Ein CMS Block besteht aus zwei Komponenten: Der CMS Block an sich und dem CMS Element das die Darstellung übernimmt. Wir können existierende Elemente verwenden um unseren CMS Block zu befüllen, in unserem Fall werden wir zwei vordefinierte Elemente verwenden: `text` und `image`.

Wir werden einen CMS Block mit Text und Bild Inhalt erstellen, daher wird er der Text-Image Kategorie zugeordnet. Wir erstellen daher unseren Ordner unter `src/Resources/app/administration/src/module/sw-cms/blocks/text-image/bow-tut-box`. In diesem Ordner legen wir unsere `index.js` Datei für den Block an:

```
import './component';
import './preview';

Shopware.Service('cmsService').registerCmsBlock({
    name: 'text-image-bow-tut-box',
    label: 'sw-cms.blocks.textImage.bowTutBox.label',
    category: 'text-image',
    component: 'sw-cms-block-text-image-bow-tut-box',
    previewComponent: 'sw-cms-preview-bow-tut-box',
    defaultConfig: {
        marginBottom: '20px',
        marginTop: '20px',
        marginLeft: '20px',
        marginRight: '20px',
        sizingMode: 'boxed'
    },
    slots: {
        subtitle: {
            type: 'text',
            default: {
                config: {
                    content: {
                        source: 'static',
                        value: 'Bild Untertitel'
                    }
                }
            }
        },
        image: {
            type: 'image',
            default: {
                data: {
                    media: {
                        url: '/administration/static/img/cms
/preview_plant_large.jpg'
                    }
                }
            }
        }
    }
});
```

4.1.1 component Komponent

Der `component` Unterordner beinhaltet den Komponenten der im Admin Bereich beim editieren des Inhalts angezeigt wird. Der Komponent besteht aus 3 Dateien:

- index.js Die Definitionsdatei
- sw-cms-block-bow-tut-box.html.twig Die Twig Template Datei für den Komponenten

- sw-cms-block-bow-tut-box.scss Eine optionale SCSS Datei für das Styling

In der index.js definieren wir den Komponenten an sich:

```
import template from './sw-cms-block-bow-tut-box.html.twig';
import './sw-cms-block-bow-tut-box.scss';

const { Component } = Shopware;

Component.register('sw-cms-block-text-image-bow-tut-box', {
    template
});
```

Die Twig Datei beinhaltet das Layout und die Slots für die Elemente:

```
{% block sw_cms_block_bow_tut_box %}
    <div class="sw-cms-block-bow-tut-box">
        <slot name="image"></slot>
        <div class="subtitle">
            <slot name="subtitle"></slot>
        </div>
    </div>
{% endblock %}
```

Die SCSS Datei beinhaltet das Styling für den Komponenten:

```
.sw-cms-block-bow-tut-box {
  .sw-cms-el-image {
    margin: 0 auto;
    img {
      max-height: 360px;
    }
  }
  .subtitle {
    text-align: center;
  }
}
```

4.1.2 preview Komponent

Der Preview Komponent wird in der Sidebar beim hinzufügen neuer CMS Blöcke verwendet. Er zeigt also nur eine kleine Preview für den Komponenten an. Dieser Komponent besteht aus den gleich 3 Dateien wie der `component`.

Die Index Datei importiert die anderen beiden Dateien und registriert den Komponent:

```
import template from './sw-cms-preview-bow-tut-box.html.twig';
import './sw-cms-preview-bow-tut-box.scss';

const { Component } = Shopware;

Component.register('sw-cms-preview-bow-tut-box', {
    template
});
```

Die Twig Datei beinhaltet die Placeholder:

```
{% block sw_cms_block_bow_tut_box_preview %}
    <div class="sw-cms-block-text-image-bow-tut-box">
        <div class="block-title content-block">
            <div class="image" />
            <div class="text" />
        </div>
    </div>
{% endblock %}
```

Und die SCSS Datein beinhaltet ein bisschen Styling um die Vorschau gut verständlich zu machen:

```
.sw-cms-block-text-image-bow-tut-box {
  .image {
    height: 150px;
    background-size: 30% auto;
    background-color: rgba(0,0,0, 0.1);
    background-image: url('data:image/svg+xml;utf8,%3Csvg%20viewBox%3D%
220%200%20282.69%20228%22%20xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F2000%
2Fsvg%22%3E%3Ccircle%20fill%3D%22%23A5A5A5%22%20cx%3D%22115.3%22%20cy%
3D%2235.75%22%20r%3D%2235.75%22%2F%3E%3Cpath%20fill%3D%22%23A5A5A5%22%
20d%3D%22M188.7%2C228h-81.34c-10.27%2C0-16.24-11.86-10.28-20.41l38.69-
55.48l42.65-61.2%20c5.03-7.22%2C15.53-7.22%2C20.56%2C0l42.64%2C61.17l38.
7%2C55.51c5.96%2C8.55-0.02%2C20.4-10.28%2C20.4H188.7z%22%2F%3E%3Cpath%
20fill%3D%22%23A5A5A5%22%20d%3D%22M2.48%2C206.79l55.44-78.81c4.27-6.07%
2C12.64-7.54%2C18.72-3.29l112.83%2C78.81%20c10.8%2C7.54%2C5.46%2C24.51-
7.71%2C24.51l-168.27%2C0C2.58%2C228-3.8%2C215.71%2C2.48%2C206.79z%22%2F%
3E%3C%2Fsvg%3E');
    background-repeat: no-repeat;
    background-position: 50%;
  }

  .text {
    margin: 8px auto;
    height: 16px;
    width: 75%;
    background: #666;
    border-radius: 8px;
  }
}
```

4.1.3 Admin JS einbinden

Die JS Scripts für den Adminbereich werden ähnlich wie im Storefront in einer `main.js` Datei gebündelt. Diese Datei befindet sich in `src/Resources/app/administration/src/main.js`. Wir müssen dort nur unseren Block importieren:

```
import './module/sw-cms/blocks/text-image/bow-tut-box';
```

4.2 Storefront Komponent

Shopware versucht automatisch unseren Komponenten zu laden, dazu verwendet es den Blocknamen, in unserem Fall ist das `cms-block-text-image-bow-tut-box`. Die Datei wird unter `src/Resources/views/storefront/block` angelegt:

```
{% block block_bow_tut_box %}
    <div class="bow-tut-box-block">
        {% block block_bow_tut_box_image %}
            {% set element = block.slots.getSlot('image') %}
            <div class="cms-element-bow-tut-box-image" data-cms-element-
id="{{ element.id }}">
                {% sw_include "@Storefront/storefront/element/cms-
element-" ~ element.type ~ ".html.twig" ignore missing %}
            </div>
        {% endblock %}

        {% block block_bow_tut_box_subtitle %}
            {% set element = block.slots.getSlot('subtitle') %}
            <div class="cms-element-bow-tut-box-subtitle" data-cms-
element-id="{{ element.id }}">
                {% sw_include "@Storefront/storefront/element/cms-
element-" ~ element.type ~ ".html.twig" ignore missing %}
            </div>
        {% endblock %}
    </div>
{% endblock %}
```

Da wir Standard Elemente verwendet haben (`image` und `text`) können wir diese hier verwenden. Für das Styling fügen wir einfach etwas SCSS zu unserem Storefront SCSS hinzu:

```
.cms-block-text-image-bow-tut-box {
  .bow-tut-box-block {
    margin: 0 auto;
    border: 1px solid #666;
    border-radius: 4px;

    .cms-element-bow-tut-box-image img {
      border-radius: 4px 4px 0 0;
    }

    .cms-element-bow-tut-box-subtitle {
      text-align: center;
      padding: 8px 0;
    }
  }
}
```

5. Externe Links

- Tutorial Plugin Bitbucket
- Shopware 6 Developer Seite