Workbook

SmartBoxes und SmartElements in inCMS

Grundlagen, Möglichkeiten, Grenzen, JSON-Editor und Praxisbeispiele

Ziel dieses Workbooks

Dieses Workbook soll eine belastbare Arbeitsgrundlage geben: Was leisten SmartBoxes und SmartElements in inCMS? Was ist mit reinem JSON möglich? Wo beginnen die Grenzen? Und wie entwickelt man eigene Elemente systematisch, ohne sich in Versuch und Irrtum zu verlieren?

Stand: Mai 2026. Quellenbasis: SwissMadeMarketing-Supportartikel, Transkript von drei Tutorial-Videos sowie die im Gespräch geprüften SmartBox-/SmartElement-JSON-Dateien.

Inhalt

1. Kurzfassung: Was man sich merken sollte

SmartBoxes und SmartElements sind in inCMS keine vollwertigen serverseitigen Module. Sie sind vor allem HTML-/CSS-/JavaScript-Bausteine mit einem konfigurierbaren Editor. Genau darin liegt ihre Stärke: Man kann wiederverwendbare Gestaltungselemente bauen, die für Redakteure bedienbar bleiben.

Die wichtigste Unterscheidung: Eine SmartBox umschließt vorhandenen Inhalt über den Platzhalter %CONTENT%. Ein SmartElement erzeugt dagegen selbst konkreten Inhalt, zum Beispiel ein Video, eine Anzeige, einen Button oder eine manuelle Bildwand.

Merksatz	Bedeutung für die Praxis
SmartBox = Container	Sie eignet sich für Hintergründe, Abstände, volle Breite, Rahmen, farbige Bereiche und das Umgestalten vorhandener inCMS-Module.
SmartElement = Baustein	Es eignet sich für wiederverwendbare Elemente mit festen Feldern: Video, Karte, Bildbutton, Anzeige, manuelle Galerie, Icons, Hinweisbox.
Editor-JSON = Bedienoberfläche	Im JSON wird festgelegt, welche Felder der Redakteur sieht: Farbe, Zahl, Auswahl, Datei, Text, Checkbox usw.
HTML/CSS/JS = technische Ausgabe	Die eigentliche Gestaltung und Funktion entsteht im HTML-, CSS- und JavaScript-Block.
Keine Ordner-Logik belegt	Aus den geprüften Unterlagen ergibt sich kein belastbarer Hinweis, dass ein SmartElement direkt einen Filebrowser-Ordner auslesen und über alle Bilder iterieren kann.
Browser testen	Sobald JavaScript, CSS Columns, Sticky-Elemente oder Speziallayouts beteiligt sind, muss in Chrome, Safari, Firefox und mobil getestet werden.

Praktische Schlussfolgerung

SmartBoxes sind sehr gut für kontrollierte Gestaltung. Sie ersetzen aber nicht automatisch ein Galerie-, Datenbank- oder PHP-Modul. Für dynamische Inhalte bleibt meist das bestehende inCMS-Modul zuständig; die SmartBox gestaltet es nur um.

2. Begriffe: SmartBox, SmartElement, Wrapper

Die Tutorials stellen SmartBoxes als Weiterentwicklung der früheren Wrapper dar. Der Grundgedanke bleibt: Ein Bereich wird mit eigenem HTML, CSS und optional JavaScript umgeben oder als eigener Baustein ausgegeben. Im ersten Tutorial wird erklärt, dass SmartBoxes früher Wrappers hießen und jetzt über einen SmartBox-Button leichter an bestehenden Modulen bearbeitet werden können.

Die offizielle Supportstruktur ordnet die Artikel im Entwicklerbereich ein: Grundlagen, Full-Width-Container und alle JSON-Editor-Optionen. Das zeigt bereits, dass SmartBoxes sowohl ein Anwenderwerkzeug als auch ein Entwicklerwerkzeug sind.

Begriff	Arbeitsdefinition	Typischer Einsatz
SmartBox	Ein Container, der um vorhandenen Inhalt gelegt wird. Enthält im HTML meist %CONTENT%.	Full-Width-Bereich, Hintergrundbild, farbiger Abschnitt, Innenabstand, Rahmen, visuelle Kapselung.
SmartElement	Ein eigenständiger Inhaltstyp, dessen HTML meist direkt Platzhalter wie {IMG}, {TITLE} oder {LINK} enthält.	Video, Anzeigenbox, Sticky-Button, Icon, manuell gepflegte Bilderwand.
Wrapper	Ältere Bezeichnung bzw. technische Grundidee hinter SmartBoxes.	Historischer Begriff, in JSON-Dateien taucht weiterhin oft wrappers auf.
Editor-Definition	JSON-Konfiguration, die die sichtbaren Felder im SmartBox-Editor erzeugt.	Felder für Farben, Zahlen, Dateien, Texte, Auswahlfelder, Checkboxen.
Definition	Block mit html, css, js und editor.	Technischer Kern jeder exportierten SmartBox-/SmartElement-Datei.

3. Einsatz im inCMS-Editor: global, lokal, Full Width

Die Tutorials unterscheiden zwei Ebenen: Wird eine SmartBox in den Einstellungen bearbeitet, wirkt diese Definition als Vorlage für die Website. Wird sie direkt an einem Modul auf der Seite verändert, betrifft die Änderung nur dieses konkrete Modul. Diese Trennung ist wichtig, weil sie erklärt, warum identisch benannte SmartBoxes lokal anders aussehen können.

Für Full-Width-Seiten wird im Tutorial empfohlen, zuerst gestalterisch zu planen: Farbpalette, Bildmaterial, Hintergründe und Abstände. Die technische SmartBox allein macht noch keine gute Seite. Sie schafft nur den Rahmen.

Arbeitsschritt	Ort im System	Wirkung
SmartBoxes aktivieren	Einstellungen / Allgemeine Einstellungen / Experten-Modus SmartBoxes	Ermöglicht das Anlegen und Bearbeiten von SmartBox-Definitionen.
SmartBox-Vorlagen bearbeiten	Einstellungen / SmartBoxes bzw. Design / SmartBoxes	Ändert die Vorlage bzw. Standarddefinition der SmartBox.
SmartBox lokal bearbeiten	Direkt am Inhaltsmodul über SmartBox-Button	Ändert nur die konkrete Instanz auf dieser Seite.

Full Width aktivieren	Checkbox oder Klasse, meist über {CLS} und html:true	Fügt eine Klasse wie fullwidth hinzu, wenn die Checkbox aktiv ist.
Inhalt verschachteln	Modul in SmartBox ziehen oder SmartBox als Inhaltsmodul einsetzen	Der Inhalt landet an der Stelle von %CONTENT%.
Vorschau prüfen	Seitenvorschau und echte Browseransicht	Zeigt, ob Abstände, Hintergründe, Bilder und Responsivität funktionieren.

4. Grundaufbau einer JSON-Datei

Die exportierten Dateien folgen einem wiederkehrenden Muster. Die äußerste Ebene enthält in den geprüften Beispielen ein Feld wrappers mit einem oder mehreren Einträgen. Jeder Eintrag enthält einen Namen, optionale Standardwerte und eine definition. Die definition enthält typischerweise html, css, js und editor.

```
{
  "wrappers": [
    {
      "name": "Name des Elements",
      "definition": {
        "html": "<div class=\"beispiel\">%CONTENT%</div>",
        "css": ".beispiel { padding: 20px; }",
        "js": "",
        "editor": "{ ... Editor-JSON als String ... }"
      },
      "_locked": 0,
      "beispiel": {
        "color": "#000000"
      }
    }
  ]
}
```

Auffällig ist: Der editor-Block ist innerhalb der Datei selbst als Zeichenkette gespeichert. Das bedeutet: Anführungszeichen, Zeilenumbrüche und Kommas müssen besonders sauber sein. JSON-Fehler führen dazu, dass die Editor-Definition nicht gespeichert werden kann.

Warum kleine Fehler große Wirkung haben

Ein fehlendes Komma, ein falsch geschriebenes inputType oder eine falsche Klammer reicht. Dann erscheint entweder der Editor nicht, die Datei lässt sich nicht speichern oder ein Feld wird nicht ausgegeben. Im Entwickler-Tutorial wird genau diese Fehleranfälligkeit angesprochen.

5. Der Editor-JSON: Klassen, Eigenschaften, Felder

Die offizielle Dokumentation beschreibt die Grundsyntax so: Es gibt eine CSS-Klasse, darunter CSS-Eigenschaften, darunter die Konfiguration des Editorfeldes. Diese Felder erzeugen die Bedienoberfläche für den Kunden oder Redakteur.

```
{
  "colorsection": {
    "background-color": {
      "label": "Hintergrundfarbe",
      "inputType": "colorfield",
      "value": "#ffffff"
    }
  }
}
```

Damit wird für die Klasse .colorsection ein Editorfeld angelegt. Der dort gewählte Wert wird als background-color in die CSS-Ausgabe geschrieben.

inputType / Option	Funktion	Bemerkung
text	Einfaches Textfeld	Laut Dokumentation als einfacher Eingabetyp verfügbar.
textfield	Textfeld in vielen Beispiel-Dateien	In den hochgeladenen Beispielen häufig genutzt; praktisch für Titel, URL, kurze Texte.
textarea	Mehrzeiliger Text	In Smart Ads für Content genutzt.
numberfield	Nur numerische Eingabe	Standardausgabe ist px, wenn outputType nicht gesetzt ist.
colorfield	Farbwähler	Mit showAlpha:true kann Transparenz erlaubt werden.
filefield	Dateiauswahl	Für Bilder, Videos, Hintergrundbilder, Favicons usw. geeignet.
imagefilefield	Bildauswahl	In Beispielen für einzelne Bilder belegt; mit maxX/maxY nutzbar.
linkfield	Linkauswahl	Für interne/externe Links komfortabler als reines Textfeld.
select	Auswahlfeld	options enthält Wert links und Beschriftung rechts.
checkbox	Ja/Nein-Schalter	checkedValue und uncheckedValue definieren die Ausgabe.
fields	Gruppiert mehrere Unterfelder	Gut für Padding, Margin, Border, Positionen.
label	Beschriftung im Editor	Macht CSS-Felder für Redakteure verständlich.
_label	Feldsatz-Überschrift	Gliedert den Editor, z. B. SmartBox, Background, Border.
html:true	Wert wird in HTML-Platzhalter geschrieben	Nicht als CSS-Regel, sondern z. B. in {CLS}, {IMG}, {LINK}.

outputType	Ausgabeformat bei numberfield	Mögliche Werte laut Doku: %, em oder "" für 1:1-Ausgabe.
allowBlank	Feld darf leer sein	Nützlich, wenn Bild oder Farbe optional sind.
value	Standardwert	Wird verwendet, wenn noch nichts gewählt wurde.

6. HTML-Platzhalter und html:true

Der Unterschied zwischen normaler CSS-Ausgabe und HTML-Platzhalter ist zentral. Ohne html:true wird der Wert einer Eigenschaft als CSS auf die genannte Klasse geschrieben. Mit html:true sucht inCMS im HTML nach einem Platzhalter in geschweiften Klammern und ersetzt diesen dort.

```
HTML:
<a href="{LINK}" class="button">{TEXT}</a>

Editor:
{
  "button": {
    "link": {
      "label": "Link",
      "inputType": "linkfield",
      "html": true
    },
    "text": {
      "label": "Button-Text",
      "inputType": "textfield",
      "html": true
    }
  }
}
```

Das Prinzip wird bei Checkboxen besonders nützlich. Eine Checkbox kann bei Aktivierung eine CSS-Klasse in das HTML schreiben und bei Deaktivierung nichts ausgeben.

```
HTML:
<div class="smart-box smart-main {CLS}">%CONTENT%</div>

Editor:
"cls": {
  "label": "Full Width",
  "inputType": "checkbox",
  "checkedValue": "fullwidth",
  "uncheckedValue": "",
  "html": true
}
```

Typischer Stolperstein

Bei Zahlfeldern darf man nicht unbedacht px doppelt erzeugen. Wenn inCMS aus einem numberfield bereits 5px macht, darf CSS nicht nochmals calc(var(--gap) * 1px) daraus machen. Besser ist entweder direkte CSS-Ausgabe als px oder bei HTML-Platzhaltern outputType:"" und dann {GAP}px selbst setzen.

7. CSS, Spezifität und Vererbung

SmartBoxes leben und sterben mit sauberem CSS. Ein häufiger Fehler: Man denkt, eine Regel greift nicht, obwohl sie nur von einer spezifischeren inCMS-Regel überschrieben wird. Im Entwickler-Tutorial wird am Beispiel SmartFlexi gezeigt, dass Formular-Labels ihre Farbe nicht übernehmen, wenn eine stärkere vorhandene Regel sie auf Schwarz setzt. Die Lösung: gezieltere Selektoren oder color: inherit ! important für bestimmte Elemente.

```
.smartflexi label,
.smartflexi button,
.smartflexi p,
.smartflexi h1,
.smartflexi h2,
.smartflexi h3,
.smartflexi h4,
.smartflexi h5 {
  color: inherit !important;
}
```

Problem	Ursache	Saubere Gegenmaßnahme
Farbe wird nicht übernommen	Kind-Element hat eigene, stärkere CSS-Regel.	Selektor präzisieren, Vererbung bewusst setzen, notfalls !important gezielt einsetzen.
Abstand wird ignoriert	numberfield gibt px aus, CSS erwartet aber eine reine Zahl.	outputType:"" setzen oder var direkt mit px nutzen.
Bild erscheint nicht	Falscher Feldtyp, falscher Platzhalter oder html:true fehlt.	Quelltext prüfen: Steht wirklich ein src-Wert im HTML?
SmartBox wirkt nur teilweise	Sie umschließt nicht das richtige Element oder das Ziel liegt tiefer.	DOM-Struktur in Entwicklertools prüfen.
Globaler Stil überschreibt lokalen Stil	Reihenfolge oder Spezifität der CSS-Dateien.	Selektoren bewusst enger fassen oder CSS im globalen Projekt-CSS setzen.
Layout unterscheidet sich in Firefox	Browser unterscheiden sich bei CSS Columns, Breaks und einigen Layoutdetails.	Firefox testen; bei echten Masonry-Layouts ggf. JavaScript verwenden.

8. JavaScript in SmartElements

Die JSON-Struktur erlaubt einen js-Block. Damit können SmartElements einfache clientseitige Logik ausführen: leere Felder ausblenden, Bilder nach dem Laden verteilen, Animationen ergänzen oder Klassen setzen. JavaScript ist aber kein Ersatz für serverseitige Funktionen. Es kann nur mit dem arbeiten, was im fertigen HTML bereits vorhanden ist oder was über öffentlich erreichbare Dateien nachgeladen werden darf.

Geeignet für JavaScript	Nicht geeignet für JavaScript innerhalb normaler SmartElements
Leere Platzhalter ausblenden	Geschützte inCMS-Daten serverseitig auslesen

DOM-Elemente nach dem Laden umsortieren	Direkter Zugriff auf Filebrowser-Ordner ohne API
Einfaches Masonry auf vorhandenen Bildfeldern	PHP-Logik, Datenbankabfragen, Benutzerrechte prüfen
Interaktionen wie Tabs, Akkordeon, Toggle	Zuverlässige SEO-Ausgabe für nachträglich geladene Inhalte
Klassen setzen, Größen messen, Browser-Fallbacks	Sicherheitsrelevante Funktionen

Praxisregel

Je wichtiger Inhalt und SEO sind, desto weniger sollte er erst nachträglich per JavaScript entstehen. Für visuelle Ergänzungen ist JS in Ordnung. Für tragende Inhaltslogik ist ein echtes Modul oder vorhandenes inCMS-Modul besser.

9. Was geht - und was nicht geht

Die folgende Tabelle ist als Entscheidungsgrundlage gedacht. Sie trennt zwischen nachweislich gut geeignet, möglich mit Vorsicht und vermutlich nicht mit reinem SmartElement-JSON lösbar.

Anforderung	Bewertung	Begründung
Full-Width-Bereich mit Hintergrundfarbe	Gut geeignet	Standardfall der SmartBox.
Full-Width-Bereich mit Hintergrundbild und Parallax-Option	Gut geeignet	In Standard-SmartBox-Beispielen vorgesehen.
Innenabstand, Außenabstand, Rahmen, Rundung konfigurierbar machen	Gut geeignet	Über numberfield, fields, colorfield, select gut abbildbar.
Vorhandenes Inhaltsmodul optisch kapseln	Gut geeignet	%CONTENT% macht genau das.
Ein einzelnes Bild, Video, Icon oder Button ausgeben	Gut geeignet	filefield/imagefilefield/linkfield/html:true reichen.
Tabs/Akkordeon/kleine Interaktion	Gut geeignet mit Test	HTML/CSS/JS möglich, aber browserübergreifend testen.
Manuelle Galerie mit 10-40 Bildfeldern	Möglich	Einzelne filefield-Felder funktionieren; Pflegeaufwand beachten.
Normales inCMS-Galeriemodul umstylen	Möglich, aber fehleranfällig	Abhängig von DOM-Struktur, CSS-Spezifität und Ladefolge.
Alle Bilder eines Ordners automatisch ausgeben	Mit den geprüften Mitteln nicht belegt	Kein Ordnerfeld, keine Loop-Syntax, kein Serverzugriff in den Beispielen erkennbar.

Datenbankabfragen, PHP-Logik, Rechteprüfung	Nicht Aufgabe normaler SmartElements	Dafür braucht es Module, API oder systemseitige Erweiterung.
Beliebige komplexe App im SmartElement	Nur begrenzt sinnvoll	Technisch teilweise möglich, aber Wartung, Datenschutz, Performance und Editorbedienung werden problematisch.

10. Entwicklungsworkflow

Der zuverlässigste Weg ist nicht: großes Element bauen und hoffen. Der zuverlässigste Weg ist: klein beginnen, eine Sache testen, dann erweitern. Gerade weil der Editor-JSON empfindlich auf Syntaxfehler reagiert, muss man schrittweise arbeiten.

1. **1. Ziel klären:** Ist es ein Container oder ein Inhaltselement? Muss vorhandener Content umschlossen werden oder erzeugt das Element eigenen Content?
2. **2. Minimal-HTML schreiben:** Nur die notwendige Struktur. Bei SmartBoxen %CONTENT% nicht vergessen.
3. **3. Statische CSS-Version testen:** Erst ohne Editorfelder prüfen, ob HTML und CSS grundsätzlich funktionieren.
4. **4. Editorfelder ergänzen:** Pro Schritt nur ein bis zwei Felder ergänzen. Nach jedem Schritt speichern und testen.
5. **5. Platzhalter prüfen:** Bei html:true im Seitenquelltext prüfen, ob {IMG}, {LINK}, {CLS} ersetzt wurden.
6. **6. Zahlen und Einheiten prüfen:** numberfield gibt standardmäßig px aus. Für HTML-Platzhalter ggf. outputType:"" verwenden.
7. **7. Browser prüfen:** Chrome, Safari, Firefox, Smartphone. Besonderes Augenmerk auf Spalten, Sticky, JS und Bildlayouts.
8. **8. Quelltext auswerten:** Wenn etwas nicht greift: DOM-Struktur und aktive CSS-Regeln in den Entwicklertools prüfen.
9. **9. Robustheit erhöhen:** Leere Felder ausblenden, alt-Texte erlauben, sinnvolle Standardwerte setzen.
10. **10. Dokumentieren:** Name, Zweck, Felder, bekannte Grenzen und Teststatus in der JSON-Datei oder separater Notiz dokumentieren.

11. Praxisbeispiele

11.1 Minimaler Farbcontainer

```
HTML:
<div class="colorsection">%CONTENT%</div>

CSS:
.colorsection { padding: 30px; }

Editor:
{
  "colorsection": {
    "background-color": {
      "label": "Hintergrundfarbe",
      "inputType": "colorfield",
      "value": "#f5f1ee"
    }
```

```
  }
}
```

Dieses Beispiel ist ideal zum Lernen: Eine Klasse, eine Eigenschaft, ein Feld. Wenn das funktioniert, hat man das Grundprinzip verstanden.

11.2 Standard-SmartBox für Full Width

Die Standard-SmartBox nutzt zwei Ebenen: einen äußeren Container für Hintergrund und Breite, einen inneren Container für den eigentlichen Inhalt. Die Full-Width-Option wird über eine Checkbox in eine Klasse geschrieben.

```
HTML:
<div class="smart-box smart-main {CLS}">
  <div class="smart-box-container smartbox-main-inner">
    %CONTENT%
  </div>
</div>
```

Teil	Aufgabe
smart-main	Hintergrundfarbe, Hintergrundbild, Bildposition, Wiederholung, Parallax/Attachment.
smartbox-main-inner	Innenabstand und ggf. Breite des Inhalts.
{CLS}	Platzhalter für eine Klasse wie fullwidth.
%CONTENT%	Position, an der der eingeschlossene inCMS-Inhalt ausgegeben wird.

11.3 SmartFlexi

SmartFlexi ist ein gutes Lernbeispiel, weil es zeigt, wie ein vorhandenes Element optisch kontrollierbarer gemacht wird: Schriftfarbe, Hintergrundfarbe, Rahmen, Rundung und Padding werden über Editorfelder steuerbar.

```
HTML:
<div class="smartflexi {EQUAL}">%CONTENT%</div>

CSS-Auszug:
.smartflexi label,
.smartflexi button,
.smartflexi p,
.smartflexi h1,
.smartflexi h2,
.smartflexi h3,
.smartflexi h4,
.smartflexi h5 {
  color: inherit !important;
}
```

11.4 Smart Video

Smart Video zeigt die typische SmartElement-Logik: Es gibt keinen %CONTENT%-Bereich. Stattdessen erzeugt das Element ein eigenes Video-HTML mit Platzhaltern für Video-URL, Poster und Optionen.

```
<video class="smart-video" {AUTOPLAY} {CONTROLS} {LOOP} {MUTED} poster="{POSTER}">
  <source src="{VIDEO-URL}" type="video/mp4">
  Your browser does not support the video tag.
</video>
```

11.5 Manuelle Masonry-Bildwand

Die im Gespräch entwickelte Masonry-Bildwand ist ein Beispiel für die Grenze zwischen SmartElement und dynamischer Galerie. Da kein Ordner-Iterator belegt ist, ist die robuste Lösung eine manuell gepflegte Bildauswahl mit mehreren filefield-Feldern. Für echte lückenarme Masonry-Darstellung kann JavaScript die Bilder nach dem Laden auf zwei Spalten verteilen.

Variante	Vorteil	Nachteil
CSS Columns	Einfach, lückenarm, wenig Code.	Browserunterschiede möglich, vor allem Firefox testen.
CSS Grid	Sehr stabil und sauber.	Erzeugt Lücken bei unterschiedlich hohen Bildern.
JavaScript Balanced Columns	Lückenarm und browserrobuster als CSS Columns.	Mehr Code, muss nach Bildladung und bei Resize sauber laufen.
inCMS-Galerie überschreiben	Ordnerauswahl bleibt erhalten.	Sehr abhängig vom erzeugten HTML und den bestehenden Galerie-CSS-Regeln.

12. Fehlerdiagnose und Prüflisten

12.1 JSON-Check

- Alle geschweiften Klammern geschlossen?
- Nach jedem Eintrag auf gleicher Ebene ein Komma - außer nach dem letzten?
- Anführungszeichen sauber gesetzt?
- editor-JSON gültig, obwohl es als String gespeichert ist?
- inputType exakt richtig geschrieben?
- Klassenname im Editor entspricht einer Klasse im HTML?
- Bei html:true existiert der passende Platzhalter im HTML?

12.2 CSS-Check

- Ist die gewünschte Regel im Browser überhaupt geladen?
- Wird sie durchgestrichen, also überschrieben?
- Ist der Selektor spezifisch genug?
- Liegt ein Inline-Style vor, der stärker ist?
- Ist der Wert syntaktisch gültig, z. B. nicht calc(5px * 1px)?
- Wird die Regel im Bearbeitungsmodus anders behandelt als in der Vorschau?
- Gibt es Unterschiede zwischen Chrome, Safari und Firefox?

12.3 HTML-/Platzhalter-Check

- Im Quelltext prüfen: Wurde {IMG} durch eine echte Bild-URL ersetzt?
- Bei Links prüfen: Enthält href eine sinnvolle URL oder bleibt es leer?
- Bei Checkbox-Klassen prüfen: Wird die Klasse nur gesetzt, wenn aktiv?
- Sind leere optionale Felder sichtbar? Falls ja: per CSS oder JS ausblenden.
- Sind Alt-Texte möglich und gepflegt?
- Sind Dateipfade relativ/absolut so, wie inCMS sie erwartet?

Symptom	Wahrscheinliche Ursache	Schnelltest
Editorfeld erscheint nicht	JSON-Fehler, falsches inputType, falsche Verschachtelung.	Ein Feld entfernen, speichern, Schritt für Schritt wieder ergänzen.
Wert erscheint im Editor, aber wirkt nicht	Falsche Klasse/Eigenschaft oder CSS wird überschrieben.	Entwicklertools: aktive CSS-Regel prüfen.
Platzhalter bleibt sichtbar	html:true fehlt oder Feldname passt nicht.	HTML und Editor-Namen vergleichen.
Bild lädt nicht	filefield-Wert leer, falscher Platzhalter, src leer.	Seitenquelltext: img src prüfen.
Abstand stimmt nicht	Einheit doppelt oder fehlt.	Quelltext: style-Wert kontrollieren.
Element funktioniert nur in Chrome	Browserabhängiges CSS/JS.	Firefox/Safari testen, Fallback bauen.

13. Arbeitsblätter

13.1 Planungsblatt für ein neues SmartElement

Frage	Notiz
Name des Elements	
SmartBox oder SmartElement?	
Welches Problem soll es lösen?	
Welche Inhalte erzeugt oder umschließt es?	
Welche Felder braucht der Redakteur wirklich?	
Welche Standardwerte sind sinnvoll?	
Welche Felder dürfen leer bleiben?	
Welche Browser müssen getestet werden?	

Welche Grenzen/Restriktionen müssen dokumentiert werden?	

13.2 Felddefinition planen

Feldname	inputType	html:true?	Standardwert	Pflicht?	Bemerkung

13.3 Testprotokoll

Test	Chrome	Safari	Firefox	Mobil	Notiz
Layout korrekt					
Abstände korrekt					
Bilder/Dateien laden					
Links funktionieren					
Leere Felder sauber ausgeblendet					
Editorwerte werden übernommen					

Quelltext plausibel					

14. Quellen und Anhang

14.1 Quellenbasis

Quelle	Hinweis / URL
SwissMadeMarketing Support: SmartBox - Grundlagen	https://support.swissmademarketing.com/de/support/solutions/articles/3000071041-smartbox-grundlagen
SwissMadeMarketing Support: SmartBox - die Lösung für Container auf volle Breite	https://support.swissmademarketing.com/de/support/solutions/articles/3000046411-smartbox-die-l%C3%B6sung-f%C3%BCr-container-auf-volle-breite
SwissMadeMarketing Support: SmartBox - alle JSON Editor Optionen	https://support.swissmademarketing.com/de/support/solutions/articles/3000046413-smartbox-alle-json-editor-optionen
SwissMadeMarketing Support: Import and export SmartBoxes	https://support.swissmademarketing.com/en/support/solutions/articles/3000100887-import-and-export-smartboxes
Transkript von drei Tutorial-Videos	Vom Nutzer bereitgestellt, Datei: Eingefügter Text(6).txt
Geprüfte JSON-Beispiele	Working Hours, Smart Video, AIO Fields, SmartFlexi, Smart Ads, Sticky Chat Image, Smart Tabs, Editor Styles, Masonry-Varianten

14.2 Auswertung der geprüften JSON-Beispiele

Datei	Name	Typ nach Ausgabe	Umfang
SmartBox_Editor-Styles_v2.0.json	! Editor Styles	SmartElement	HTML 22 / CSS 1475 / JS 2224
SmartBox_Masonry_Image_Grid_2_Columns_v1.0.json	Masonry Image Grid 2 Columns	SmartBox	HTML 51 / CSS 596 / JS 0
SmartBox_Smart Tabs_v1.0.json	Smart Tabs	SmartBox	HTML 52 / CSS 753 / JS 0
SmartBox_SmartFlexi_v2.1.json	SmartFlexi	SmartBox	HTML 47 / CSS 165 / JS 0
SmartBox_inCMS_Gallery_Masonry_2_Columns_v1.1.json	inCMS Gallery Masonry 2 Columns	SmartBox	HTML 54 / CSS 1701 / JS 0
SmartBox_inCMS_Gallery_Masonry_2_Columns_v1.2.json	inCMS Gallery Masonry 2 Columns	SmartBox	HTML 94 / CSS 2871 / JS 2593
SmartBox_inCMS_Gallery_M	inCMS Gallery Masonry 2	SmartBox	HTML 2207 / CSS 0 / JS 0

asonry_2_Columns_v1.3_INLINE.json	Columns Inline		
SmartElement_All In One (AIO) Fields_v1.0.json	AIO Fields	SmartElement	HTML 508 / CSS 0 / JS 0
SmartElement_Manual_Masonry_Image_Grid_20_v1.0.json	Manual Masonry Image Grid 20 Images	SmartElement	HTML 3835 / CSS 747 / JS 335
SmartElement_Manual_Masonry_Image_Grid_20_v1.1.json	Manual Masonry Image Grid 20	SmartElement	HTML 3850 / CSS 1136 / JS 613
SmartElement_Manual_Masonry_Image_Grid_20_v1.2.json	Manual Masonry Image Grid 20 v1.2	SmartElement	HTML 3897 / CSS 1124 / JS 643
SmartElement_Manual_Masonry_Image_Grid_20_v1.4_JS_BALANCED.json	Manual Masonry Image Grid 20 JS Balanced	SmartElement	HTML 4294 / CSS 1338 / JS 3039
SmartElement_Smart Ads_v1.0.json	Smart Ads	SmartElement	HTML 406 / CSS 606 / JS 0
SmartElement_Smart Video_v1.0(1).json	Smart Video	SmartElement	HTML 240 / CSS 51 / JS 0
SmartElement_Smart Video_v1.0.json	Smart Video	SmartElement	HTML 240 / CSS 51 / JS 0
SmartElement_Sticky Chat Image_v1.0.json	Sticky Chat Image	SmartElement	HTML 69 / CSS 420 / JS 0
SmartElement_Working Hours_v1.0.json	Working Hours	SmartElement	HTML 64 / CSS 150 / JS 0

14.3 Beispiel: robuster Umgang mit numberfield und Einheiten

```
Problematisch, wenn --gap bereits 5px enthält:
column-gap: calc(var(--gap, 5) * 1px);

Robuster, wenn inCMS px ausgibt:
column-gap: var(--gap, 5px);

Robuster bei HTML-Platzhalter:
Editor: "gap": { "inputType":"numberfield", "outputType":"", "html":true }
HTML:  <div style="--gap:{GAP}px"> ... </div>
CSS:   column-gap: var(--gap);
```

14.4 Beispiel: Entscheidungsbaum

- Brauche ich vorhandenen Content im Inneren? -> SmartBox mit %CONTENT%.
- Erzeuge ich ein festes Element mit Feldern? -> SmartElement ohne %CONTENT%.
- Brauche ich eine Liste aus Dateien, Produkten, Artikeln oder Datenbank? -> Erst prüfen, ob ein vorhandenes inCMS-Modul das liefert.
- Brauche ich nur Gestaltung? -> SmartBox.
- Brauche ich serverseitige Logik? -> eigenes Modul, API oder Supportanfrage; nicht normales SmartElement-JSON.
- Brauche ich nur eine kuratierte Auswahl? -> Mehrere filefield/imagefilefield-Felder können genügen.