ItkCriticalCss

Entwicklerdokumentation

André Fischer

30.03.2026

Inhaltsverzeichnis

1 Einordnung

- **Plugin-Klasse:** `Itk\\CriticalCss\\ItkCriticalCss`
- **Shopware:** `shopware/core` ^6.6.0, PHP ^8.1
- **Admin-Label:** „Critical CSS Generator“

Symfony / Shopware kurz: Das Plugin nutzt überwiegend **Event Subscriber** (**EventSubscriberInterface**, Methode `getSubscribedEvents()`). Shopware ruft sie auf, wenn ein Ereignis eintritt – z. B. **StorefrontRenderEvent** kurz vor dem Rendern einer Storefront-Seite (Parameter für Twig setzen) oder `ConsoleEvents::TERMINATE` nach Ende eines Konsolenbefehls. Eine **Twig-Extension** (`twig.extension`-Tag) ergänzt Filter/Funktionen für Templates. Du musst keine Symfony-Route-Controller schreiben, um die Hauptlogik zu verstehen.

---

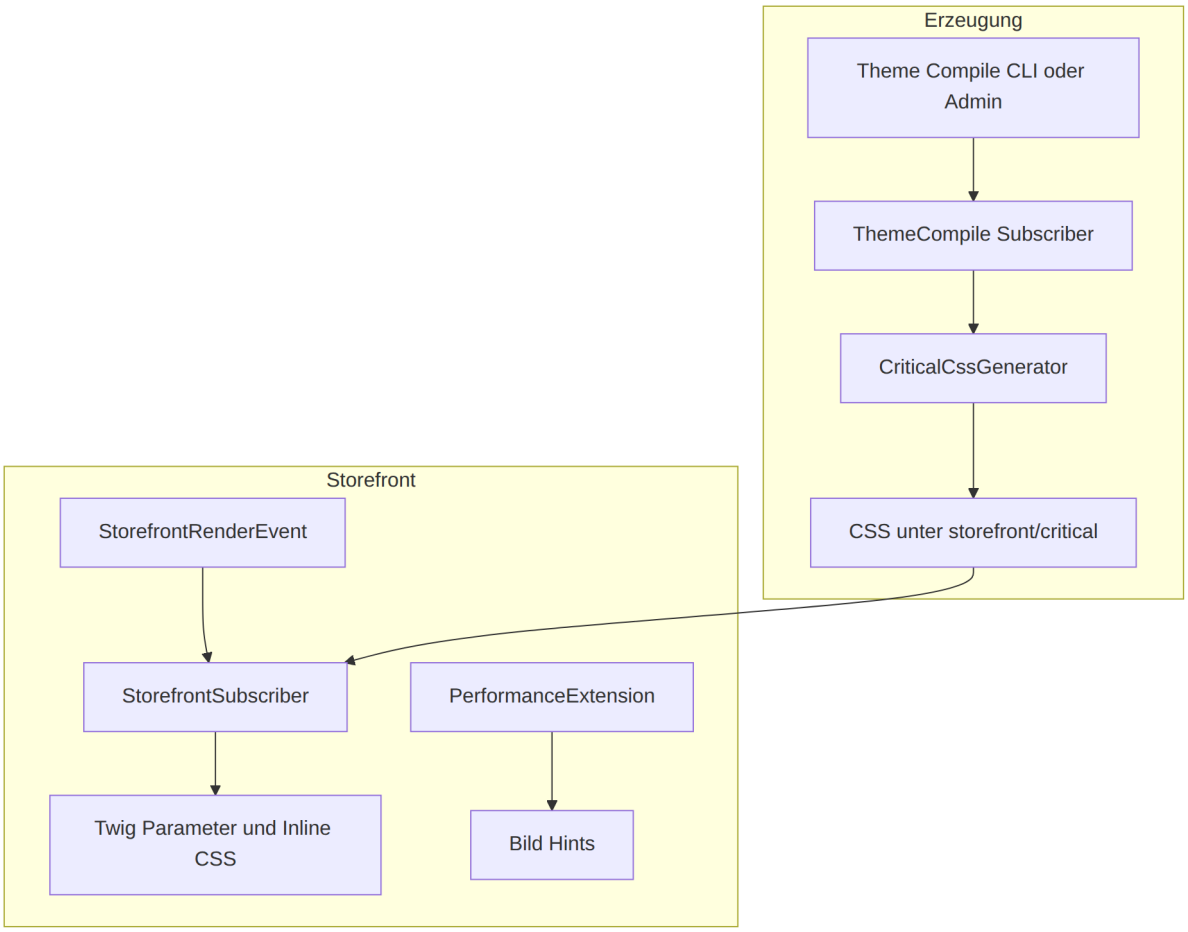
2 Architektur-Überblick

PHP-Namespace: `Itk\\CriticalCss\\…` (Plugin-Ordner **ItkCriticalCss**).
SystemConfig-Präfix: `ItkCriticalCss.config.*`.

Beim Storefront-Rendering setzt das Plugin Twig-Parameter und **inline Critical CSS** (Inhalt aus **CSS-Dateien** im Plugin), Resource Hints, Theme-Preload, LCP-Preload. Bild-Hints (**loading**, **fetchpriority**) kommen über die **Twig-Extension PerformanceExtension**.

Generierte Dateien liegen unter
`custom/plugins/ItkCriticalCss/src/Resources/views/storefront/critical/`
(z. B. `critical_<salesChannelId>.css`, `critical_product_<salesChannelId>.css`, Fallbacks `critical.css`, `cls-fallback.css`).

2.1 Diagramm: Erzeugung und Auslieferung



---

3 Konfiguration

Alle Keys unter `ItkCriticalCss.config.*`:

Key	Typ	Default	Beschreibung
injectCriticalCss	bool	true	Hauptschalter
serviceUrl	string	–	Externer Critical-CSS-Generator
preconnectUrls	string	–	Kommagetrennte Preconnect-Domains

Key	Typ	Default	Beschreibung
fontPreconnectUrls	string	–	Font-Preconnect-Domains (crossorigin)
lcpPreloadUrl	string	–	Exakte URL des LCP-Bildes
asyncStylesheets	bool	false	Theme-CSS async laden
preloadThemeCss	bool	true	Theme-CSS preloaden
lcpImageOptimization	bool	false	eager + fetchpriority an LCP-Kandidaten
thumbnailLazyDefault	bool	false	lazy als Default für Bilder ohne loading-Attribut
autoClsPrevention	bool	true	CLS-Korrekturregeln zum Critical CSS anhängen

---

4 `services.xml` und Laufzeitmodell

Das Plugin ist technisch schlank, aber sehr klar aufgeteilt:

- **CriticalCssGenerator** erzeugt Dateien
- **StorefrontSubscriber** liest und injiziert diese Dateien beim Rendern
- **ThemeCompileSubscriber** und **ThemeCompileBackendSubscriber** stoßen die Generierung nach Theme-Kompilierung an
- **PerformanceExtension** ergänzt Twig-Funktionen für Bild- und Performance-Hinweise
- die Commands sind reine Einstiegspunkte in dieselbe Generator-/Diagnose-Logik

Wichtig aus `services.xml`:

Service	Abhängigkeiten	Rolle
CriticalCssGenerator	project_dir, **logger**, **SystemConfigService**, http_client, sales_channel_domain.repository, product.repository	erzeugt Homepage- und Produkt-CSS
StorefrontSubscriber	project_dir, **logger**, **SystemConfigService**, **AbstractThemePathBuilder**, request_stack, **ClsPreventionService**	liest Dateien, hängt Zusatzregeln an, setzt Twig-Parameter

Service	Abhängigkeiten	Rolle
ThemeCompileSubscriber	**CriticalCssGenerator**, **AbstractAvailableThemeProvider**, **logger**	reagiert auf CLI-Theme-Compile
ThemeCompileBackendSubscriber	**CriticalCssGenerator**, **logger**	reagiert auf Compile aus dem Admin
DiagnoseCommand	project_dir, **SystemConfigService**, Theme-Provider/Builder, Repositories	erstellt Betriebsdiagnose

Für Symfony-Einsteiger wichtig: Es gibt hier **keinen Controller** als Hauptdrehscheibe. Das Verhalten entsteht fast komplett aus **Services + Subscriber + Twig**.

---

5 CriticalCssGenerator

Klasse: `Itk\\CriticalCss\\Service\\CriticalCssGenerator`
Pfad: `custom/plugins/ItkCriticalCss/src/Service/CriticalCssGenerator.php`

generate(?string $salesChannelId, ?string $overrideUrl): string

1. **Homepage-URL:** aus erster `sales_channel_domain` zum Kanal oder **overrideUrl**.
2. **generateForUrl**:
   - Wenn `ItkCriticalCss.config.serviceUrl` **gesetzt** → **generateViaService**: Symfony `http_client` GET, Query **url=**, Antwort muss `text/css` sein, wird in Zieldatei geschrieben.
   - Wenn **leer** → `assertNodeAvailable()` (**node –version**), `assertCriticalAvailable()` (`node_modules/critical` im Plugin-Root), dann **Process**: **node /src/Resources/app/storefront/scripts/generate-critical.js** (Timeout 300 s).
3. Zusätzlich: **Produktdetail-URL** über `product.repository` (aktives Produkt mit Cover, Sichtbarkeit Kanal, kein Kind) + Domain → zweite Datei `critical_product_<salesChannelId>.css`. Fehler hier → Log-Warnung, Homepage-Datei bleibt gültig.

Ausgabepfade: getOutputPath() → …/views/storefront/critical/{critical|critical_|critical_product_}.css.

Wichtig: `getPluginRoot()` ist **hart** …/`custom/plugins/ItkCriticalCss` relativ zu `kernel.project_dir` – bei abweichender Projektstruktur bricht die Pfadlogik.

---

6 ClsPreventionService

ClsPreventionService generiert CSS-Regeln, die häufige Layout-Shifts verhindern. Das umfasst typischerweise:

- Explizite Dimensionierung für Bilder ohne **width**/**height**-Attribute
- Platzhalter-Regeln für Elemente die erst nach JS-Load sichtbar werden
- Aspect-Ratio-Boxen für Responsive-Bilder

Diese Regeln werden, wenn **autoClsPrevention = true**, an das geladene Critical CSS angehängt, bevor es in den HTML-Head injiziert wird.

---

7 StorefrontSubscriber

`Itk\\CriticalCss\\Subscriber\\StorefrontSubscriber` auf **StorefrontRenderEvent**.

Ablauf (vereinfacht):

1. Setzt immer Parameter für **Resource Hints**, **LCP-Produkt-URL**, **Theme-Stylesheet-Infos**, **Config** (für Twig).
2. Erkennt **Produktdetail** (**ProductPage** oder Route `frontend.detail.page`).
3. Wenn **injectCriticalCss** aus → **itkCriticalCssInline** leer, Return.
4. Sucht eine **existierende CSS-Datei** in fester Reihenfolge:
   - Produktseite: `critical_product_<scId>.css` → `critical_product.css` → `critical_<scId>.css` → `critical.css` → `cls-fallback.css`
   - Sonst: `critical_<scId>.css` → `critical.css` → `cls-fallback.css`
5. Liest Dateiinhalt; ignoriert Platzhalter **/* ITK_CRITICAL_PLACEHOLDER */**.
6. `ClsPreventionService::generate($page, $route)` liefert zusätzliches CSS; **Deduplizierung** gegenüber Critical-CSS; **Minify** (einfacher String-Reducer im Subscriber).

7. Setzt Twig-Parameter **itkCriticalCssInline**, **itkCriticalPreloadThemeCss**, etc.

Die Ausgabe erfolgt über **Twig-Templates** des Plugins (Block-Erweiterungen), nicht durch nachträgliches Parsen des finalen HTML.

Twig: `Itk\\CriticalCss\\Twig\\PerformanceExtension` – registriert als `twig.extension` für Bild-Attribute u. Ä.

---

8 ThemeCompileSubscriber / ThemeCompileBackendSubscriber

- **ThemeCompileSubscriber**: lauscht auf `ConsoleEvents::TERMINATE`. Nur wenn der beendete Befehl exakt `theme:compile` heißt **und** der Exit-Code **Erfolg** ist, wird für die gefilterten Sales Channels nacheinander `CriticalCssGenerator::generate($salesChannelId)` aufgerufen (Filter aus `--only`/`--skip`/`--only-themes`/`--skip-themes` des Compile-Aufrufs).
- **ThemeCompileBackendSubscriber**: **ThemeCompilerConcatenatedStylesEvent** – ruft `generate($event->getSalesChannelId())` auf, **aber** bricht sofort ab, wenn **PHP_SAPI === 'cli'** (damit es nicht doppelt mit dem CLI-Subscriber kollidiert). Deckt damit **Admin-Theme-Compile** ab.

Sofern der Generator auf der ausführenden Maschine funktioniert (Node + **npm install** im Plugin-Root oder erreichbare **serviceUrl**), entstehen bzw. aktualisieren sich die CSS-Dateien unter …/`views/storefront/critical/`.

Für den Betrieb heißt das:

- CLI-Compile und Admin-Compile nutzen bewusst **unterschiedliche Trigger**
- der Backend-Subscriber schützt sich gegen Doppelverarbeitung unter **CLI**
- wenn nach Admin-Compile nichts passiert, darfst du nicht nur den CLI-Subscriber prüfen
- wenn nach CLI-Compile nichts passiert, ist oft nicht der Subscriber kaputt, sondern die Generator-Umgebung auf dem Server

---

9 CLI-Commands

9.1 itk:critical:generate

```
php bin/console itk:critical:generate --sales-channel-id=<uuid>
```

Ruft intern `CriticalCssGenerator::generate()` auf. **Ohne** `--sales-channel-id` bricht die Generierung mit einer Exception ab (**resolveHomepageUrl** verlangt eine gültige Kanal-ID). Pro Kanal also **ein Aufruf** – oder ihr nutzt den Ablauf nach **bin/console theme:compile**, der die Kanäle in einer Schleife abarbeitet.

Je nach Konfiguration: **serviceUrl** gesetzt → HTTP-GET zum Dienst; **leer** → lokal **Node** + `generate-critical.js` (siehe **CriticalCssGenerator**).

9.2 itk:critical:diagnose

```
php bin/console itk:critical:diagnose
```

Gibt u. a. aus: Projekt/PHP-Umgebung, **existierende `.css`-Dateien** im Verzeichnis …/`storefront/critical/`, Theme-/Sales-Channel-Infos, relevante **SystemConfig**-Werte. Optional `--sales-channel-id` / **-s** für eine gefilterte Ausgabe. Nicht jede Zeile der alten Stichwortliste muss zutreffen – als **Support-Dump** den kompletten Text verwenden.

9.3 itk:critical-css:test

```
php bin/console itk:critical-css:test
```

Einfacher Connectivity-Test, ob der externe Service antwortet.

---

10 Twig-Templates

Das Plugin hat eigene Templates unter `Resources/views/storefront/`, die Shopware-Blöcke erweitern. Die relevanten Blöcke sind im **head**:

- `base_head_meta_tags`: Dort werden Critical CSS, Preconnects und Preloads eingefügt.
- Ggf. eigene Blöcke im Body für LCP-Bild-Optimierungen.

Wenn andere Plugins in die gleichen Blöcke schreiben, kann es zu Konflikten kommen. Das ist bei Shopware-Plugins eine bekannte Herausforderung – die Priorität der Template-Erweiterungen lässt sich über den Bundle-Priority-Wert steuern.

---

11 Debugging

Critical CSS wird nicht injiziert:

1. **injectCriticalCss** in der Konfiguration ist aus? Einschalten.
2. Existieren Dateien unter …/`storefront`/`critical`/? → `itk:critical:diagnose` / Verzeichnis listen.
3. **itk:critical:generate –sales-channel-id=** ausführen (ohne ID schlägt der Generator fehl).
4. Ohne **serviceUrl**: **Node** und **npm install** im Plugin-Root; mit **serviceUrl**: URL im Browser/curl testen.
5. Log auf **„CLS-Fallback aktiv“** prüfen – dann fehlen die Hauptdateien.

Visueller Fehler nach Theme-Änderung: Das Critical CSS ist veraltet. `cache:clear` oder gezielt **itk:critical:generate –sales-channel-id=…** (pro Kanal) bzw. `theme:compile` per CLI nutzen.

Preloads erscheinen nicht im head: Browser-DevTools → Quelltext anschauen (nicht “Inspektor”, der zeigt die DOM-Ansicht nach JS-Verarbeitung). Im raw HTML prüfen ob die Preload-Tags vorhanden sind. Falls nicht, Subscriber prüfen ob er den richtigen Twig-Block erweitert.

Generierung läuft, aber Datei bleibt leer oder unbrauchbar:

1. Mit externer **serviceUrl** Response-Typ und Response-Body prüfen; der Service muss verwertbares **CSS** liefern, nicht HTML/JSON.
2. Ohne **serviceUrl** Node-Umgebung, Plugin-Root und installierte `node_modules`/`critical` prüfen.
3. Prüfen, ob die aufgelöste Homepage-/Produkt-URL im Ziel-Sales-Channel tatsächlich erreichbar ist.
4. Im Zweifel zuerst mit **Homepage-Datei** debuggen; die Produkt-Datei ist bewusst nur ein zweiter Schritt und darf separat fehlschlagen.

Unterschied zwischen lokal und Server:

Der Generator hängt an realer Laufzeitumgebung:

- Node-Version
- installierte npm-Pakete
- Erreichbarkeit der Ziel-URL
- Schreibrechte im Ausgabeordner
- abweichende Projektstruktur relativ zu `kernel.project_dir`

Deshalb ist dieses Plugin besonders anfällig für „läuft lokal, aber nicht auf Staging/Prod“-Effekte.