Task 2 – Shopware 6 Plugin

Topic: Loading CSS and JavaScript Efficiently to Increase Page Load Speed (Mobile)

I built a Shopware 6 plugin that improves PageSpeed by **inlining page-specific critical CSS**, **loading the remaining CSS asynchronously**, and **delaying non-essential tracking scripts** while keeping the core storefront JavaScript behavior intact.

Problem statement

Shopware storefronts often suffer from:

- Render-blocking CSS
- Heavy third-party tracking scripts
- Large unused CSS/JS bundles

This negatively impacts:

- First Contentful Paint (FCP)
- Largest Contentful Paint (LCP)
- Total Blocking Time (TBT)

Objective of the plugin

To improve mobile page load performance by **Eliminating render-blocking of CSS & Js**

Solution overview

The plugin introduces three core optimizations:

CSS

- Inline critical CSS (above-the-fold)
- Asynchronously load the full stylesheet

JavaScript

- Keep core JS **deferred** (Shopware default)
- Delay tracking scripts only

All implemented via:

- Twig partial injection
- Storefront**Render**Event subscriber
- System Config - Inline critical script and Enable / disable features
- Page-specific critical CSS files generated via the **critical** npm package

---

Plugin architecture

Execution flow:

- `HTTP Request`
- `↓`
- `Storefront`**`Render`**`Event`
- `↓`
- `Plugin` **`Subscriber`**
- `↓`
- `setParameter()`
- `↓`
- **`Twig rendering`** `(base.html.twig & product-detail.html.twig)`
- `↓`
- `Final HTML output`

Key design choices:

- Uses `StorefrontRenderEvent` to inject rendering data
- Passes variables using `setParameter()`
- Injects logic into **`base_head`** (always rendered)
- Uses SystemConfigService for admin configuration

Main features

1, CSS optimization (Hybrid - Text area & critical css file)

- Page-specific critical CSS:
    - Homepage / CMS pages
    - Category pages

    - Product detail pages

- Inline CSS via:
  `<style id="critical-css">...</style>`
- Async loading of remaining CSS using preload replacement

2, JavaScript optimization

- Core JS remains:
  `<script src="storefront.js" defer></script>`

- Tracking scripts delayed using:
  ```
  window.addEventListener('load', () => {
    setTimeout(loadTracking, 3000);
  });
  ```

Backend configuration

Admin can configure:

- Enable / disable plugin
- Critical CSS content per page type
- Enable / disable tracking delay
- Delay duration (seconds)

Twig integration

Files used:

- `storefront/util/css-js-optimizer.html.twig`
- `storefront/base.html.twig`
- `storefront/page/content/product-detail.html.twig`

Injection:

- `{% block base_head %}`
- `  {{ parent() }}`
- `  {% sw_include '@YourTheme/storefront/util/css-js-optimizer.html.twig' %}`
- `{% endblock %}`

Rendering comparison

Before

- `HTML → CSS download → render → JS`

After

- `HTML → render immediately`
- `CSS downloads in parallel`
- `CSS applied later`
- `JS executes after DOM`

Technical advantages

- Upgrade-safe
- No core modification
- Theme-compatible
- Page-specific optimization
- Backend configurable

Challenges solved

- Correct Twig injection across themes, Twig inheritance.
- SystemConfigService, I initially forgot to declare the constructor argument in services.xml.
