# Concept - partial caching

**Draft for ?**

a.k.a component caching, a.k.a snippet caching, a.k.a Edge Side Includes-ish.

Allow caching portions of a page. Needs research.

ARCHI-16 - Jira project doesn't exist or you don't have permission to view it. . See also Concept - Dynamic Page Caching

- Rationale
- Stories
- Things to consider
- Related concepts and issues
- Current thoughts

## Rationale

The current cache module allows caching entire pages. It optionally lets a component drive the cache by looking at headers before actually storing a page in the cache (see Cache Header Negotiation). However, if one component disables caching, that means the rest of the page will never be cached either.

If we implement Concept - Cache arbitrary objects, and perhaps MAGNOLIA-3902 - Configurable cache constraints on renderables OPEN (althoug h arguably, a boolean on the component definition could suffice), we could cache all-but-one components of a page.

## Stories

- As a developer, I can mark certain pages as uncacheable, without adding contention on the system or the cache (but obviously such page will perform less than a cached page) - this is done with ~~MGNLCACHE-57~~ - Avoid unnecessary cache locking for uncacheable entries CLOSED .
- As a developer, I can mark certain page **components** as uncacheable; this should result in **only** this component being re-rendered
- As a developer, I can mark certain page **components** as cacheable, with different rules than the containing page; this should result in everything being served from the cache in a 2-step process; a) fetch the page from cache or render it b) fill in the individual components that aren't in the cached page

## Things to consider

We need to remember the main goal of page caching, which is to completely avoid any connection to JCR while rendering the page.

A naive implementation would still disable the page-cache and would reconstruct the page from the cached-snippets + non-cached snippets. This would create at least 1 connection to JCR (for the page content itself), and perhaps more use of it to render the non-cached components.

A more advanced implementation would probably do something similar to Edge Side Includes or Varnish, i.e still cache the surrounding page.

- Quid cache flush on activation ?

## Related concepts and issues

Concept - Configurable cache constraints on renderables - MAGNOLIA-3902 - Configurable cache constraints on renderables OPEN

Concept - Cache Improvements - ~~MGNLCACHE-37~~ - Cache configuration should be simplified CLOSED

Concept - Cache arbitrary objects - ~~MGNLCACHE-55~~ - Caching arbitrary objects CLOSED

MAGNOLIA-5550 - Allow dedicated caching of areas OPEN

~~MGNLCACHE-57~~ - Avoid unnecessary cache locking for uncacheable entries CLOSED

## Current thoughts

- implement (a subset of) ESI (or something similar)
- instead of caching a page's content, we'd cache an "object" which knows about the various snippets to include and so on. Sort of like putting a pre-parsed `freemarker.template.Template` in the cache and processing it on request.
- an additional/alternative filter processes these, "fills" in the fragments

- ESI is based around the notion that fragments are represented by a path/url, and that variables/conditions are all available in the form of headers or cookies.
- Fragments themselves would still need to be cached
  - Magnolia page components could be simply configured as "cacheable" (see Concept - Configurable cache constraints on renderables and MAGNOLIA-3902 - Configurable cache constraints on renderables OPEN or the rationale above which suggests that a simple boolean might suffice)
  - Cache keys for components/fragments might (have to) be different per component.

- ESI as such doesn't help with our current implementation of personalization, since we personalize (copy) entire pages, the whole page is potentially personalized and its components can't know if they're cacheable
- ESI helps for content pools scenarios, where a dynamic component knows it's "not cacheable" or "cacheable with certain parameters".

- "Real" ESI would be useful in CDN scenarios or even with simple varnish fronts
- We'd also need implementation of an "internal" ESI-like feature, where fragments can be resolved by other means than url headers and cookies (if we know fragment X needs the user's age, bake that in the fragment's cache key)