IMAGE HANDLING explained and further optimized v2

this case potentially covers 10 HUBnext user stories

alexander.dross@boehringer-ingelheim.com
Global Capability Lead for Search

This case covers the following user stories:

1. HNGDS-1305 Rework picture tags for images
2. HNGDS-1306 Alternative images for mobile - phase 1
3. HN-3379 Determine a consistent lazy loading solution
4. HN-1480 As a site visitor I want to lazy load images to improve mobile experience
5. HN-832 Performance - off-screen images and videos
6. HN-1092 Implementation of blazy
7. HN-3281 Change all images on the GDS components to use the HTML Picture Tag
8. HN-3517 404 Error calling lazy library /libraries/lazysizes/lazysizes.min.js
9. HN-105 Performance - optimized images and videos
10. HN-828 Performance - every image has title and ALT tag

Boehringer Ingelheim

Current Case

- since we are rebuilding and optimizing our image components to add responsive/ scaled images, doesn't it make sense to take the opportunity and additionally optimize in an accessibility point of view?
- It is already mandatory that each image contains an ALT and title tag with the same value. This means images cannot be published without having the value for these tags in place.
  (see HN-828)

Current Case

- Current image components from GDS have already a description of the image – however for now it is not mandatory to fill, to not show the description on the frontend.

Available dosage strenghs - 150-mg capsule and 100-mg capsule. Not shown at actual size

Source:
https://gds.boehringer-ingelheim.com/?path=/info/core-library-components-content--xl-content

Going one step forward in an accessibility point of view

Boehringer Ingelheim

Benefits accessibility

- To go one step forward, why not making the value “description of the image” mandatory to be filled?
- By default the description appears next to the image unless checkbox marked to hide description
- This will enable screen readers to benefit from it.

Benefits SEO

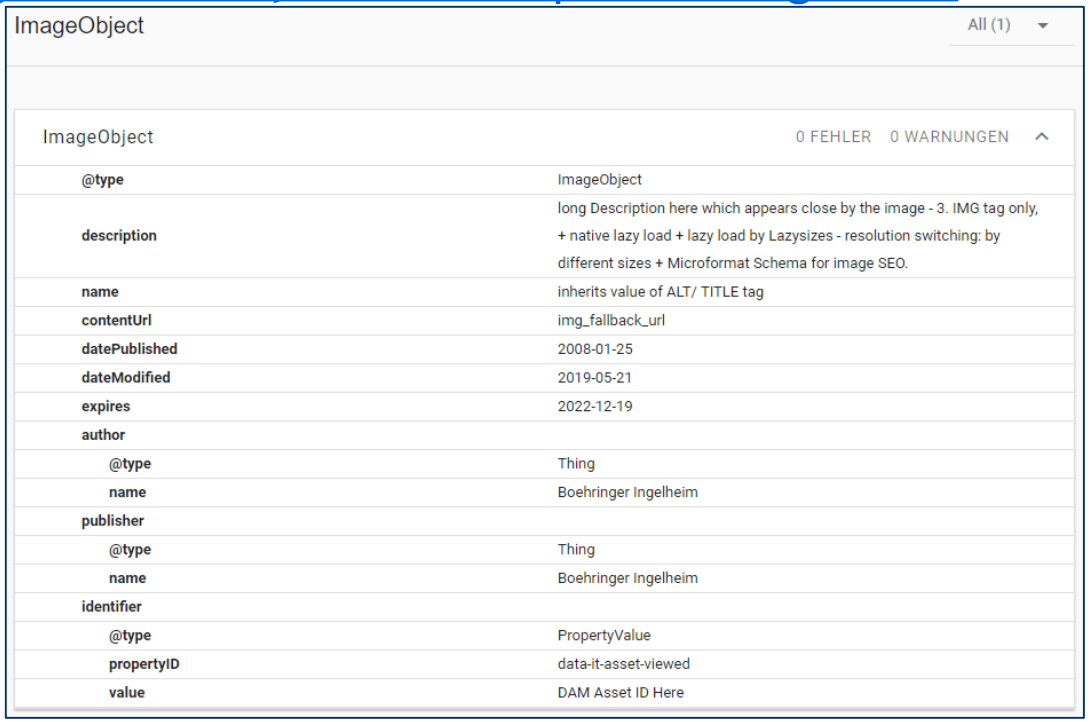
- To use the value of the “description of the image” for SEO purposes, we will use schema in Microdata format which is build-in to every image component (instead of using JSON-LD here) – we use Microdata format for header and footer navigation as well as on breadcrumbs already.

Two checkboxes in the backend of each image component:

1. checkbox, if description of the image should be hidden in the frontend, or not.
2. checkbox, if image can appear in featured snippets/ also called “position 0” (default setting is “public”) or not.
   If the checkbox is manually set = image should not appear in featured snippets as image is intended for HCPs, therefore data-nosnippet will be set, so search engines will still process the schema in Microdata format to understand what the image is about – but will not use it for featured snippets.

Why is the schema markup important?

- Search engines are not able to “see” images so it is important to name all imagery that describes the image. e.g. “boehringer-ingelheim_head_office.jpg" instead of "image1.jpg".
- However the above naming convention cannot be followed as images coming from the DAM (using the website’s root domain) include its DAM asset ID only, therefore naming of image files will look as the following*: http://hcpbi044a.boeingelacsfsf.acsitefactory.com/sites/g/files/bubxbj1876/files/aprimo/images/33c56c87-d1ea-431c-9a1a-abb3008cab4d.png
- This is why the use of the schema markup becomes even more important as search engines can use it build relations between the content of a page and images which are embedded.
- In addition, schema properties "datePublished", "dateModified", "expires" (from DAM) could be used to flush CDN cache *on asset levels* whenever a new version is released.
- Drupal tokens could be used to inherit values.



* SEO can become quite flexible and find different approaches, when other parameters are still in its bandwidth. (no image file naming convention = use of schema markup)

Use of Example Source Code

Use the following example source code to enables:

- Native lazy load
- Lazy load of images by Lazysizes, for older browsers
- webP image formats
- Responsive/ scaled images, for each break point
- Retina display optimized images
- ALT + title tags on images/ description of the image - has to come from DAM, but with option to overwrite locally in website backend
- Optimize with structured data markups in Microdata format NEW

1. **Please open https://imagehandling.drossmedia.de/**
2. **Analyze source code of 3. and 8., as the combination with Microdata should always be used.**
3. **Other versions are just intermediate steps to better understand the end result.**
4. **Download example code on https://imagehandling.drossmedia.de/image-handling.zip**
5. **See browser recording https://imagehandling.drossmedia.de/browser-recording_responsive-images.webm**

Lazy loading libraries (related to HN-1092, HN-2517)

If you're not so concerned about *how* lazy loading works under the hood and just want to pick a library and go (and there's no shame in that!), there's plenty of options to choose from. Many libraries use a markup pattern similar to the ones demonstrated in this guide. Here are some lazy loading libraries you may find useful:

- lazysizes is a full-featured lazy loading library that lazy loads images and iframes. The pattern it uses is quite similar to the code examples shown here in that it automatically binds to a `lazyload` class on `<img>` elements, and requires you to specify image URLs in `data-src` and/or `data-srcset` attributes, the contents of which are swapped into `src` and/or `srcset` attributes, respectively. It uses intersection observer (which you can polyfill), and can be extended with a number of plugins to do things like lazy load video.
- lozad.js is a super lightweight option that uses intersection observer only. As such, it's highly performant, but will need to be polyfilled before you can use it on older browsers.
- blazy is another such option that bills itself as a lightweight lazy loader (weighing in at 1.4 KB). As with lazysizes, it doesn't need any third party utilities to load, and works for IE7+. Unfortunately, it doesn't use intersection observer.
- yall.js is a library I wrote that uses IntersectionObserver and falls back to event handlers. It's compatible with IE11 and major browsers.
- If you're seeking a React-specific lazy loading library, you might consider react-lazyload. While it doesn't use intersection observer, it *does* provide a familiar method of lazy loading images for those accustomed to developing applications with React.

Each of these lazy loading libraries is well documented, with plenty of markup patterns for your various lazy loading endeavors. If you're not one to tinker, grab a library and go. It will take the least amount of effort.

Requirements for Asset Picker to serve web ready images

- compression to 72 web DPI.
- depending on type, max dimension, coming from DAM should be 1920px width (for background images), and not max 1000px width.
- convert JPG/ JPEG to progressive JPEG.
- convert PNG to progressive PNG.
- convert GIF to interlaced GIF.
- convert JPG/JPEG/PNG to webP.
- DAM provides title and ALT tag in English (content agency has to provide in local language, to overwrite).
- DAM provides file naming convention (content agency has to provide in local language, to overwrite).
- Videos in mp4 and webM format only.
- Keeping DAM asset Id for new versions of assets.
- Goal:
  We have to use ”compressed” and “scaled/ responsive images” to be served with the “correct aspect ratio”, therefore each image has to be stored the web server (Drupal media library) locally in 4-5 different dimensions for each device type. Modern web browsers will also serve images in webP format. In addition, each image must have an ALT and title tag. The DAM asset ID must be kept when new asset versions are being released, to enable proper caching per website’s CDN.

https://web.dev/serve-responsive-images/
https://gtmetrix.com/serve-scaled-images.html
https://web.dev/serve-images-with-correct-dimensions/
https://web.dev/image-aspect-ratio/
https://web.dev/use-imagemin-to-compress-images/
https://web.dev/serve-images-webp/

Process to serve web ready images to our websites

1. Go to DAM and select an image you want to use on a HUBnext/ WebPro website
2. DAM Asset Picker uploads local copy of the selected image to Drupal Media Library.
   DAM Asset Picker also exposes the following data of each image to be stored in Media Library:
   - datePublished
   - dateModified
   - Expires
   - DAM Asset ID
3. Drupal module then
   - compresses image to 72web DPI
   - converts to webP image format
   - generates progressive version of JPG/ PNG
   - resizes image into 5 different dimensions as we have 5 breakpoints:

   -> 5x JPG/ PNG, 5x @2 for Retina displays
   -> 5x webP, 5x @2 for Reting displays
   -> 1x fall-back with <img> tag
   -> 21 image versions in total, but only 1 was selected on DAM

Process to serve web ready images to our websites

4. Drupal Editor adds the following data to each image within Drupal Media Library:
   - Values for ALT/ Title tags
   - Description of the image 100-150 characters

   To ensure HN-1306 is also met - using different images per different screen sizes, e.g. when adding text to image (global OFEV) – Drupal Editor can select another image for a specific breakpoint within Drupal

   Example code available in ZIP file at https://imagehandling.drossmedia.de/