Async images rendering

Challenge: decouple image processing from page (html) rendering to speed up page delivery for pages with multiple unprocessed images e.g. backend filelist module with external storage.
In current master https://review.typo3.org/c/Packages/TYPO3.CMS/+/57646 there is already a partial solution for this, but it has a few drawbacks, e.g. that it passes request through php even for images which are already generated.

Preface

We currently face the challenge, that a page with many unprocessed images and or image variants can take a long time to render, or at worst, cannot be delivered to the client at all when the time to render exceeds timeouts PHP or of the delivering webserver.

User facing requirements

To improve the situation for users the following requirements MUST be met:

1. Improve time to first byte (TTFB) for pages with processed images, by not letting the image processing block the overall page rendering
2. Possible implementations MUST be working for TYPO3 backend requests and SHOULD be working for frontend requests
3. Each implementation of async image processing MUST NOT have security implications, specifically it MUST NOT be possible for an attacker to trigger the async image processing with arbitrary values.

Technical requirements

To improve the situation for maintenance of the product the following requirements MUST be met:

1. Async image processing MUST be part of the main processing API, so that it can applied to all places where image processing is required.
2. Since different implementations impose different requirements on the environment and come with different user facing drawbacks, it MUST be possible to switch implementations on environment level, e.g. through a global configuration option.
3. It MUST be possible to extend existing implementations and to add additional implementations or external SaaS services by the usage of a public API.
4. Code that uses image processing MUST be independent from a concrete processing implementation.

Implementation steps

Step 1

Refactor `\TYPO3\CMS\Core\Resource\Service\FileProcessingService` to allow other file processors beside `\TYPO3\CMS\Core\Resource\Processing\LocalImageProcessor` to process a file (or image).

Tim will handle that.

Step 2 (we will see if we need this step, based on experiments)

Allow to mark a `\TYPO3\CMS\Core\Resource\ProcessedFile` as *partially processed*. Doing so is done by providing a temporary public URL and adding a flag that is persisted.

Rationale:
The first is needed to let a processing implementation provide an URL that will actually process the image or points to a URL of a placeholder image. The latter is needed for asynchronous tasks to pull out all partially processed files and actually process them.

Note:
Most probably we need to know the dimensions of the final image (thumbnail), even if the image is not yet processed, so we can generate html img tag with correct dimensions set. This probably means that the calculation of the size of the scaled/cropped image need to be extracted processing code.

Step 3 Solution for BE

We get rid of the new async viewhelper (TODO find the name of the viewhelper).

Implement a `BackendPreviewProcessor` that sets the temporary URL to a `BackendUtility::getThumbnailUrl`
The first hit (image has no thumbnail processed) will return urls pointing to a TYPO3 endpoint.
This endpoint can process image and return redirect to final url (so image is retrieved by browser with correct cache headers).
The second hit (after image has been processed) will return urls to scaled image. This url will not require any further php processing.

This would solve the issue for backend usages.

Step 4 Solution for FE

We need to investigate different scenarios (cdn, remote storage, remote file processing services).
This step will result in having one or more processors which are tailored for different scenarios. E.g. One can imagine that extension which provides s3 FAL storage, also provides an s3 aware processor.

Implement further file processors, that are capable of asynchronously process files.
Examples:

- A Processor that sets the URL to the not yet existing processed file and letting the web server rewrite the URL to a file processing end point.
- A Processor that sets the URL to a placeholder image (a generic one, a lowres bitmap image representation, a svg representation, …) and a accompanied scheduler task that processes images in a background queue.
- …

----Notes from TM----

When generating response e.g. HTML TYPO3 places urls to images which contain enough information to generate image thumbnail in second request. The url would stay the same for processed and unprocessed images (it will not change over time) allowing caching of the generated html e.g. in reverse proxies. Url might look like this:
/storage/{token}/original/file/path/filename.jpg

Where token

a) contains all params required for image processing (width, height…) stored in it

```
public function generateToken(array $params)
{
    $params = json_encode($params);
    $token = hash_hmac('sha256', $params, $encryptionKey, true) . $params;
    // Base64 encode the token and transcribe the non URL-safe characters
    $token = strtr(base64_encode($token), self::BASE64_REMAP_SEARCH,
self::BASE64_REMAP_REPLACE);
    return $token;
}
```

This way thumbnail generation doesn't require a DB connection, but url might become quite long. Example implementation for Magento

https://github.com/AOEpeople/Aoe_LazyCatalogImages/blob/master/app/code/community/Aoe/LazyCatalogImages/Helper/Catalog/Image.php

b) token contains a reference to sys_file_processedfile record, which then contains all required information for image processing
This approach requires thumbnail generation script to have access to the database and perform one db insert per thumbnail still in the thread generating html.

For performance reasons, the thumbnail generation script should be called only when the image is not yet processed (is not on the disk), this could be achieved using web server rewrite.
This however might require additional configuration on the web server side (we can ship default .htaccess and nginx rules, but it might be tricky for some integrators - and maybe we should not make it default behavior but one enabled with configuration option (maybe in sitehandling)?
e.g.
https://github.com/AOEpeople/Aoe_LazyCatalogImages/blob/master/media/catalog/product/LCI/.htaccess
https://github.com/AOEpeople/Aoe_LazyCatalogImages/issues/10

Solution testing checklist (the things we should take into account when reviewing the final proposal).

- will it work when a site is behind a reverse proxy like varnish?
- will it work when images are cached/delivered by CDN?
- will it work for small installations on shared hosting (or is there a switch)?
- will it be OK for SEO?
- is there a performance penalty for uncached/BE pages with many images which are already processed?
- will it work with files stored in external storages (S3, azure blob storage,...)

Other related concepts

- Adaptive images (delivering different image sizes depending on the screen resolution) e.g https://extensions.typo3.org/extension/c1_adaptive_images/
- LQIP (low quality image placeholders)
    - SQIP (SVG-based image placeholders)

Related reviews:
https://review.typo3.org/c/Packages/TYPO3.CMS/+/60676 `[WIP] Proof of concept deferred thumbnail processing`
https://review.typo3.org/c/Packages/TYPO3.CMS/+/60644

https://review.typo3.org/c/Packages/TYPO3.CMS/+/60422/ [FEATURE] Render thumbnails in file list module deferred

File module rework (backport to 9.5 is planned)
https://github.com/TYPO3-Initiatives/digital-asset-management

Meetings

10.08.2020

Attandees: Tymoteusz Motylewski, Helmut Hummel, Tim Schreiner

Topics:

- Current state of development, especially from Helmut
- Different cases that should be taken into account
- What to work on next

Conclusions:

- We decided to throw away the patch https://review.typo3.org/c/Packages/TYPO3.CMS/+/61859 because it is not blocking the actual task of async image processing and may be needed for some cases. But we decided to deprecate the PreviewImageTask
- Helmut will keep working on backend implementation of async image processing https://review.typo3.org/c/Packages/TYPO3.CMS/+/65237
- DB table sys_fileprocessedfile will gain a new column called something like "public_processing_url" that can contain an additional url which will be used when getting the public image url. That field may contain an url to an endpoint that will do the actual image processing.
- Add new PHP class will be added to calculate the resulting image dimensions without processing the image by taking the processingConfiguration into account. That is needed to fill width and height field when creating the sys_file_processedfile entry without actual image processing
- A processor for async frontend image processing will be created that works similar to the backend implementation. To avoid problems with caches, the processor will always return the final processed image url, even if the resulting image does not exist. A change in .htaccess, nginx conf or web-config will redirect every requests to a processed image folder that does not match a file to index.php for further processing. TYPO3 then processes

the image. Some special attention should be given to documentation of this feature and to explain the updates that has to be done in .htaccess or other webserver configs