REQUEST FOR COMMENTS

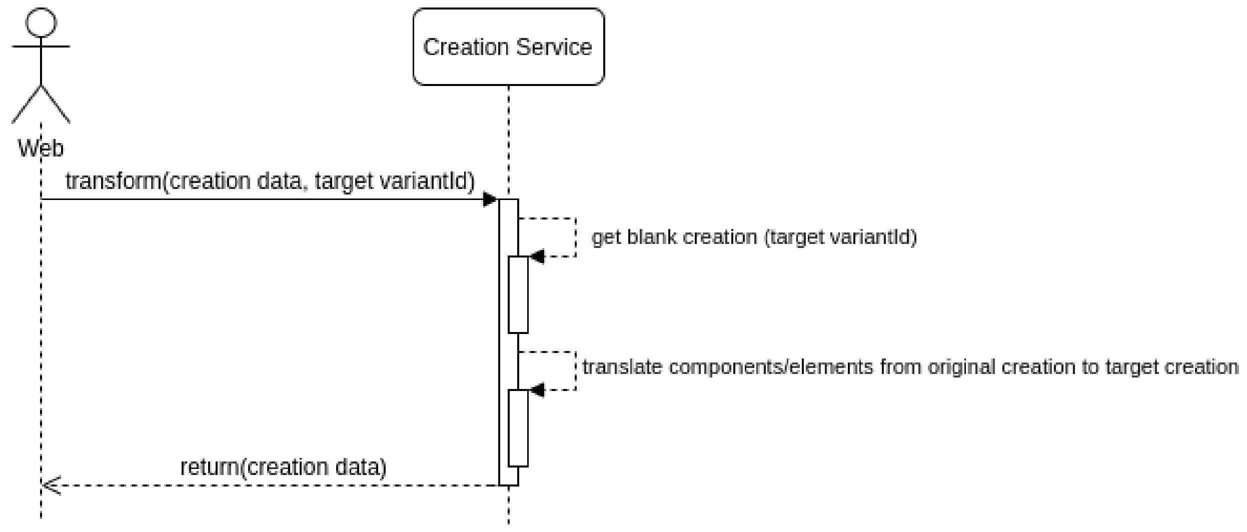
Transformation service

Problem & Context

There is a need to be able to transform a creation from variant X to variant Y, where X and Y might not belong to the same product.
Here are a few examples of possible transformations ordered by complexity, where the simplest are at the top and the most complex at the bottom:

Source	Target
Simple Canvas 30x30	Framed Canvas 30x30
Canvas 20x20	Canvas 40x40
Canvas 20x20	Canvas 50x75
A4 Classic Photobook	A3 Classic Photobook
A4 Classic Photobook	Square Spiral Book
A4 Classic Photobook	A4 Cut Out Cover Book
A4 Classic Photobook	Little Moments Book
A4 Layflat	Any other non layflat book

A “basic” transformation service is already in place, here’s a simple representation of the existing data flow:


Creation Service
Web
transform(creation data, target variantId)
get blank creation (target variantId)
translate components/elements from original creation to target creation
return(creation data)

Considered, but out of Scope

N/A

Solution

There are two areas of complexity to a transformation, one with how we translate the elements to the new surface (imagine we're going from a portrait surface to a panoramic surface) and the other is as to how we relate the components from one product to the components from another, so let's discuss these in separate sections.

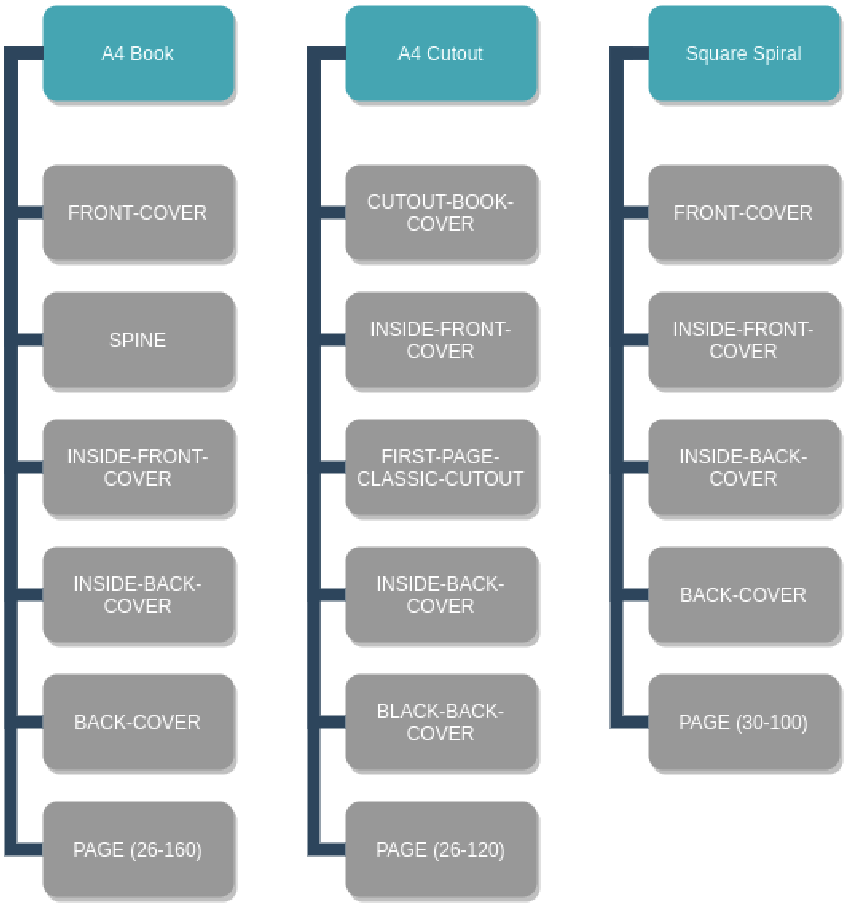
Component matching

Creations have arrays of *Components*, each of which has a *type* property which is used by the planboard data to define how these are ordered/grouped together.

Here's a representation of the components for a Standard A4 Hardcover, a Black A4 Cut Out and a Square Spiral Book as returned by the creation service:

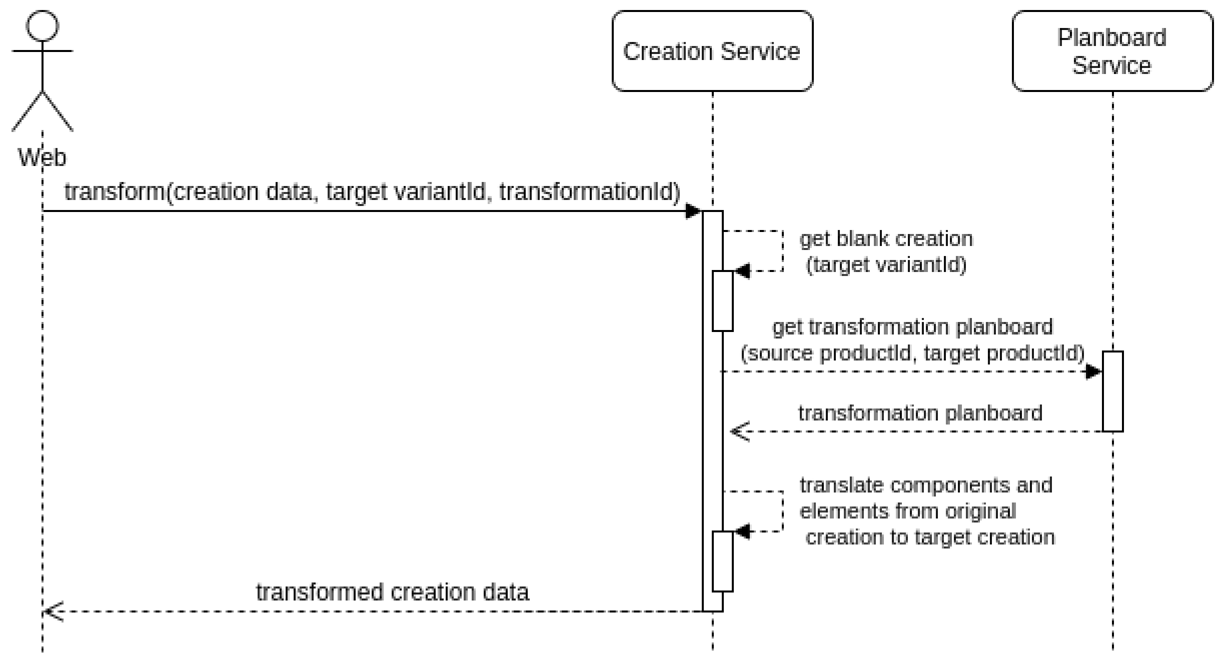
As we can see, although all these products are "Books" their composition is drastically different:

- Non matching component types
- Missing "equivalent" types
- Different minimum/maximum page counts
- Different component order
- Potential product restrictions (we don't print the covers of cutout books, so we shouldn't move content there)



Due to all the differences highlighted above, this leads us to using a concept we're already familiar with in Editor, *Planboards.*
In order to match the different components from one product to another, we should have planboards that define what each component maps to in the target **product** (these would be variant agnostic as all variants in a product share the same component types).
The format of these planboards is up for discussion further down in the document.

Here's how the updated diagram would look like:


Web
Creation Service
Planboard Service
transform(creation data, target variantId, transformationId)
get blank creation (target variantId)
get transformation planboard (source productId, target productId)
transformation planboard
translate components and elements from original creation to target creation
transformed creation data

Now with the above in mind, we need to be aware that there are still limitations with such a service, e.g:

- If going from a product with 100 max pages to one with 80 max pages, we'd have to *trim* the last ones
- If going from a product that allows a customisable spine to one without a spine, we'd have to lose the spine text
- Some transformations are nearly impossible, e.g. layflat books with full spread photos can never be truly translated to normal books which have traditional binding.

Transformation planboard structure

The planboard structure we go with is of vital importance for the success of this feature as we want it to be flexible enough to support all our use cases whilst keeping it simple enough so we avoid any unexpected *"features"*.

Here's what a transformation definition looks like to transform a standard A4 book to a little moments book:

```
{
 "components": [
   { "origin": "FRONT-COVER", "target": "LITTLE-MOMENTS-FRONT-COVER" },
   { "origin": "SPINE", "target": "SPINE" },
   { "origin": "INSIDE-FRONT-COVER", "target": "INSIDE-FRONT-COVER" },
   { "origin": "PAGE", "target": "PAGE-LITTLE-MOMENTS" },
   { "origin": "INSIDE-BACK-COVER", "target": "INSIDE-BACK-COVER" },
   { "origin": "BACK-COVER", "target": "LITTLE-MOMENTS-BACK-COVER" }
 ]
}
```

Due to how our creations are defined, these are quite re-usable across different products, in fact the transformation above could be used for transforming both the standard A4/A3 or the portrait book to a little moments book.

As such a transformation definition does not belong to a "source" productId, or a target productId and instead can be shared across multiple products depending on how the product is defined. Therefore these should live in a *transformations* folder, which is product agnostic.

```
└── transformations
     ├── 065404d4-7ad3-4c62-8cfe-4921d8d50ef0.json
     ├── 972d33c9-ce16-4469-9eea-2ae1d8bd0e40.json
     └── b43268f5-15b6-4fa5-a33f-6097dbce0943.json
```

The product will then define which transformations are possible as well as which transformation definition to use, e.g:

```
{
  "transformations": [
    {
     "target": "productId",
     "transformation": "transformationId"
    },
    ...
  ],
  ...
}
```

Whether these will link to productIds or variantIds is still TBD, although the initial preference is productIds, unless there is a strong reason to use variantIds.

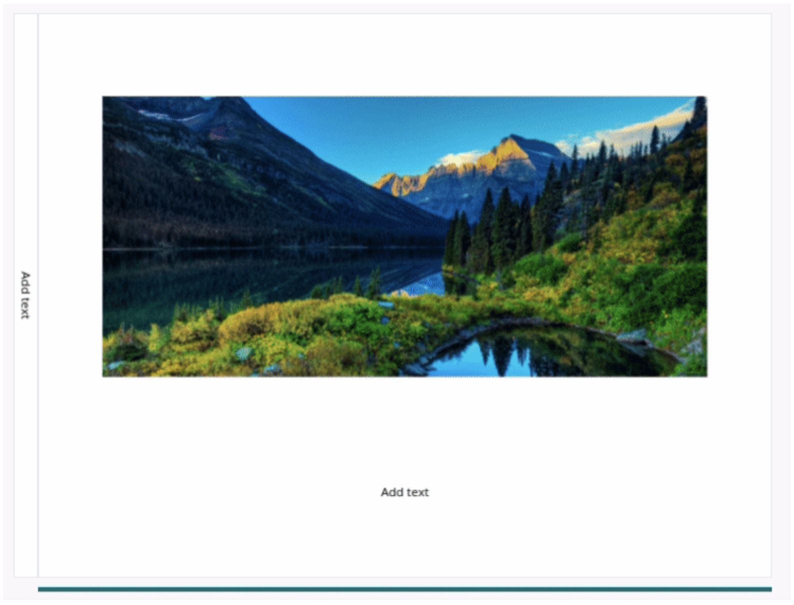
Element translation

Translating elements from one component to another of a different size, will always be performed on a “best effort” basis as achieving perfection here is nigh impossible depending on the circumstances.
With the above in mind, this section can be divided into four distinct categories, so let’s tackle each of them individually.

Photos

Photos are positioned on the page within an aperture and apertures have relative coordinates and dimensions, so those can simply be copied across.
Photo elements however, also contain a *position* property, which contains both *width* and *height* defined in *mm*. These dimensions are linked to the physical aperture size, so when moving these elements to a component with different dimensions, these values will need to be recalculated.

Another point to consider is positioning of the photo within the aperture as this will vary drastically if changing between components with very different ratios, here’s an example of how the aperture ratio affects photo positioning:



As such for MVP, we'll simply be centering the photo inside the aperture, a further improvement would be to use the AI services to detect faces or other points of interest and center the photo around those.

Decorations

Decorations are much like photos, with one major difference, **the aperture ratio always matches the decoration ratio**.
So when moving decorations to a different ratio component, we'll have to recalculate both the decoration.position.width, decoration.position.height and also the aperture.position.height and aperture.position.width.

Text

There's two main variables that control how text is displayed on the page, these are font size and the aperture width.

Font size is obvious as these will make the font bigger/smaller and in this case, we need to calculate the % difference between the current component **height** and the new component **height**, we then apply this ratio to our font size and pick the closest allowed font size from the sizes which that font supports.
We can look at integrating the component width into these calculations as well, however I'm unsure if that would give us good results.

The aperture width is relevant to how text is displayed as this is what tells the Editor to *break* text to a new line. As apertures are already positioned and sized relative to the parent component, we don't need to do anything here as when the creation is loaded on the Editor, it will recalculate the new line breaks.
It's worth pointing out though that ALL text in the creation will need have its line breaks re-calculated and at the moment, this only occurs when the customer views the component.

Backgrounds

Backgrounds are set as a property of the component, this is simply a *background* property which contains a *backgroundId*.
There is nothing wrong with how backgrounds are defined in the component, however the current structure of backgrounds in the design data poses a problem.

Currently if we have a “Leaves” background, we’ll in fact have 3 or 4 different artefacts, one for each ratio (square, landscape, portrait, panoramic). The problem here is that each of these link to a separate *background object* each one with their unique *backgroundId* and if we have the id to the “Leaves - square” background, there is no way for us to know what the id to the other corresponding ratios is.

As such we should expand the existing background schema, so that instead of a background having a single *src* property, it would instead contain a *src* property for each *ratio*. Then the Editor could pick which one to use depending on the ratio of the component.
This would mean that for the transformation purposes, we’d simply have to copy the *background* property to the new component.

Architecture

- *What does this service or feature look like? Please draw diagrams where possible (Google drawings, Lucidchart, photos of whiteboards/pad paper…)*
- *Why does it look like that?*
- *Consider the service/feature boundaries; what business capability is this service responsible for?*
- *What data does it own?*

Dependencies & Integration

How does it interact with existing services/functionality/components, both upstream and downstream? Consider both existing and planned software and services.

Interfaces

The interface exposed will be the same as the current MVP service exposes.

Infrastructure

This would be a standard Lambda function.

Scale & Performance

Lambda.

Reliability

N/A

Redundancy

N/A

Monitoring & Instrumentation

N/A

Failure Scenarios

N/A

Security

N/A

Privacy

N/A

Operational Implications

This service will be owned and monitored by the Editor team.

Future Directions

There are some ongoing talks regarding *transformations to a different Theme*, once we have more concrete requirements for this, we should look to extend the transformation endpoint and potentially re-use some of our element translation logic.

These transformations will need to be offered to customers (on the Editor I assume?), as such we'll need a place to define which transformations are possible.

Rollout

The existing transform MVP already handles image elements and image apertures, with this in mind, I'd suggest the following order of works:

1. Introduce support for simple transformations across different sized variants of the same product (Canvas)
2. Introduce support for transforming decorations & text (without accounting for rotation)
3. Introduce *simple* product transformation planboards (switch to variants on different simple products)
4. Introduce complex product transformation planboards (same ratio products)
5. Improve background asset definition, so we can support background transformations across components with different ratios
6. Introduce complex transformations across different ratio products
7. Improve *element* translations by factoring rotation into the calculations
8. Layflats ?

The layflats transformation may be pushed up/down depending on how we want to handle these transformations, i.e. *"if transforming a full spread photo to a normal book, should we simply apply this to a single page, or should we try to split the photo across two pages?"*, the former would be quite simple to achieve, while the latter will require some custom calculations to account for all the different layout options. With this said though, supporting *layflat -> different sized layflat* transformations should be quite straightforward and could be done in **step 3.**

Risks & Open Questions

N/A

Alternative Approaches

If you considered and rejected some alternative approaches, describe them. Someone reading this design might think one of these options was intuitively more obvious and it would help to explain why we are not following it.