As XYZ I want to have a service for our shop, which takes a list of image files, renders them in different sizes and stores the rendered image files.

Background

Image name format (SKU): <DesignerId>-<DesignId>-<ProductType><Orientation>-<Size>.jpg

DesignerID: Integer
DesignID: Integer
Product Types: 100 - *Poster*, 200 - *Framed Art*, 300 - *Canvas*, 400 - *Acrylic*, 500 - *AluDibond*
Orientations: L - *Landscape*, P - *Portrait*, X - *Square*
Sizes

- Portrait/landscape sizes are: 20x30, 40x60, 60x90, 80x120, 100x150
- Square sized are: 20x20, 30x30, 50x50, 70x70, 100x100

(the lists of ProductTypeIds, Orientations and Sizes are only subsets - the full lists are stored in the Database)

Sample image file names:

- 1-2-100P-80x120.jpg
- 1-3-100P-60x90.jpg
- 23-2-300X-70x70.jpg

Acceptance Criteria

- The service should be able to receive a list of image files
    - The entry point to the service should be
      *def run(files: **SomeCollection**[File]) : **AsyncExecution[ResultType]**, where*
        - ***SomeCollection** represents an iterable type, which can be a standard scala Iterable, Seq, Iterator, Stream, TraversableOnce, akka Source, etc.*
        - ***AsyncExecution[ResultType]*** represents an asynchronous aggregated result of computations. You are free to use scala Future, scalaz Tasks, Reactive Streams or a stateless future representation.
- These image files should be rendered to a different set of sizes and uploaded to S3
- Each file will go through a series of processing steps on the web server depending on the filename (see above). The filename reflects the SKU pattern (Stock Keeping Unit) of a wall art design webshop e.g.: 1-2-100P-80x120.jpg.
- For every given file the following potentially long running processing steps should be triggered on the server:
    1. move originally given file to a new unique tmp directory
    2. based on the size information given in the SKU render all possible smaller (and equal) versions of this image using ImageMagick. There will be an IMService that implements def scale(input: File, size: Size): ***AsyncExecution[File]***. This will actually be the long running process. The implementation of “def scale(...)” should be stubbed out as a TODO.
    3. upload all rendered files to S3 (should be stubbed out too)
    4. delete all locally used files
    5. return sequence of URL strings of the rendered images on S3
- **Everything that makes sense to be done concurrently, should be done concurrently**
- **Functional approach is encouraged. Consider the separation of side effects from the execution logic.**