

PicPro

PicPro

CAB432 Assignment 2

Matthias Eder

11378093

Eric Zhang

10319191

23 October 2022

Contents

[Introduction 2](#_Toc117692766)

[Purpose & description 2](#_Toc117692767)

[Services used 2](#_Toc117692768)

[Sharp Node.js API (v0.31.1) 2](#_Toc117692769)

[Use cases 2](#_Toc117692770)

[US 1 2](#_Toc117692771)

[US 2 3](#_Toc117692772)

[US n 3](#_Toc117692773)

[Technical breakdown 3](#_Toc117692774)

[Architecture 3](#_Toc117692775)

[Client / server demarcation of responsibilities 5](#_Toc117692776)

[Architecture Diagram 5](#_Toc117692777)

[Context Diagram 6](#_Toc117692778)

[Response filtering / data object correlation 6](#_Toc117692779)

[Scaling and Performance 8](#_Toc117692780)

[Test plan 9](#_Toc117692781)

[Difficulties / Exclusions / unresolved & persistent errors 10](#_Toc117692782)

[Extensions (Optional) 10](#_Toc117692783)

[User guide 10](#_Toc117692784)

[References 10](#_Toc117692785)

[Appendices 11](#_Toc117692786)

## Introduction

### Purpose & description

The PicPro Image transformer allows users to alter images using various predefined options, such as changing the filetype format, resizing the image, or adding effects like blur and greyscale. It allows new and transformed images to be stored for users that wish to use the same image multiple times, along with saving presets for transformations so that users don’t need to fill in the transformation values over and over if they wish to use the same transformation over multiple images.

Graphical user interface, application

Description automatically generated

### Services used

#### Sharp Node.js API (v0.31.1)

The Sharp API accepts and reads an image file input to allow for various transformations to be done, such as filetype format, resizing the image, or adding effects like blur and greyscale etc.

Endpoint: N/A as it is called in Node.js

Docs: <https://sharp.pixelplumbing.com/>

## Use cases

*Once again, there is no set number of use cases required, but the application will generally need at least one or two good choices as otherwise you won’t be able to generate any decent load. The basic format is provided below, and you should fill in the role and then the action and the good result that follows, just as you did in Assignment 1. Underneath the formal statement of the user story, you can then tell us how you have implemented this service – a high level description talking about services that have been used and how these relate to the scaling and persistence of the application. You should then use screenshots to illustrate the process, but this need not be very detailed.*

#### US 1

|  |  |
| --- | --- |
| As a | Graphic designer |
| I want | To transform images like changing the format type or adding filters |
| So that | I can manipulate images to suit my design |

#### US 2

|  |  |
| --- | --- |
| As a | Graphic designer |
| I want | Save previously uploaded images |
| So that | I don’t need to reupload the same image and can access it from the service with ease |

#### US n

|  |  |
| --- | --- |
| As a | Graphic designer |
| I want | Save previous transformation presets/values |
| So that | I can apply the same transformation to multiple images without having to fill in the values every time |

## Technical breakdown

### Architecture

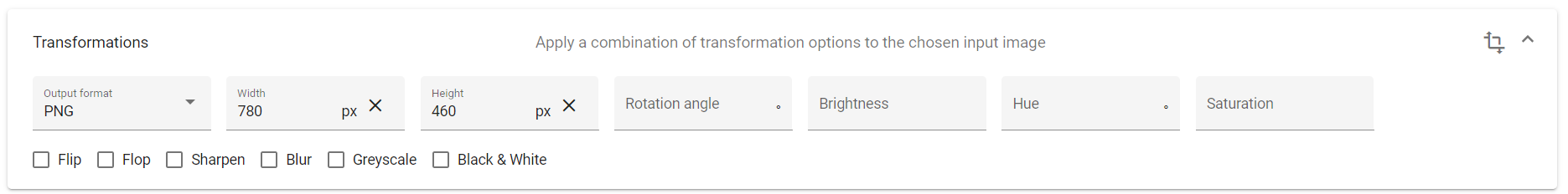
*Explain how your system operates, making it clear how data flows around the system through requests and responses, and the appearance of scaling and persistence within the architecture. In this report it is not necessary to discuss in detail the effect of these choices. Here we just want you to document the architecture and to tell us how it works.*

*The architecture for this system is split into three sections: Image, Transformations, and Presets*

Graphical user interface, application

Description automatically generated

The Image section allows users to either upload your own image or choose from a set of stored images. Uploading an image creates a POST request to store them in an s3 bucket as a Buffer object. The keys of stored images are retrieved with a GET request so users can select which one to fetch. (Which uses the original filenames as keys, like “original.png”) Once an image is chosen, a POST request is made to fetch the image data from s3, which is converted to base64 URL data and sent as a JSON response along with the filename key and metadata using the sharp API. This gets re-rendered and displayed by the frontend to display to the user



{

name: 'blank',

transformation: {

outputType: 'JPEG',

flip: false,

flop: false,

sharpen: true,

blur: true,

greyscale: false,

blackwhite: false,

width: 5,

height: 5,

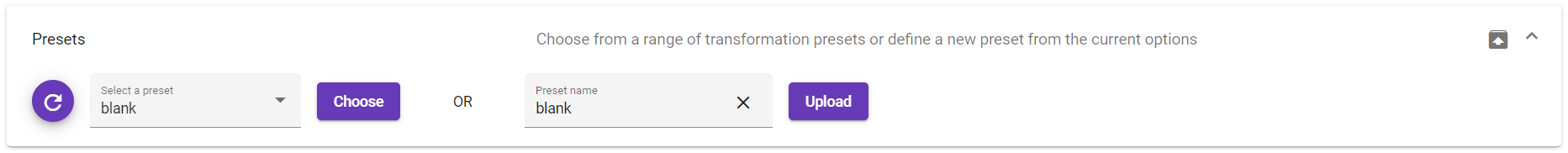
rotationAngle: 5,

brightness: 5

}

}

The Transformation sections allows users to select how they want to alter the chosen image. The values adjusted here are stored in a JSON request that gets sent alongside the data of your chosen image. Referring to the code example to the left, the transformation JSON is separated into 2 parts: “name” which just stores a unique key for the transformation that can be used later, and “transformation” that stores the values. The checkboxes are stored as Boolean values to identify if they will be applied to the image or not, and the text values get stored string and number values as they control transformations that can be adjusted or have multiple options

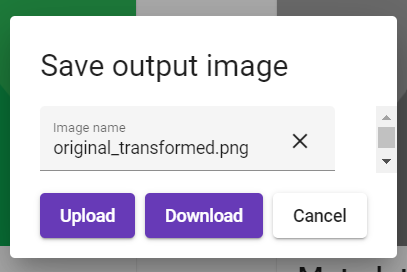


The Presets sections allows you to store transformations you previously used. Uploading a preset takes the JSON request of the currently filled in transformations from the previous section and stores them using a unique name of your choosing. This request is sent using POST that stores the preset JSON to an Elasticache cluster using Redis. Any previously stored presets have their keys retrieved like with stored images with a GET request, displaying the names of all the presets the user can choose. Once chosen, a POST fetch request is made to retrieve the desired preset, which gets sent to the frontend to be re-rendered and used to replace the values of the Transformation section.

Graphical user interface, application

Description automatically generated

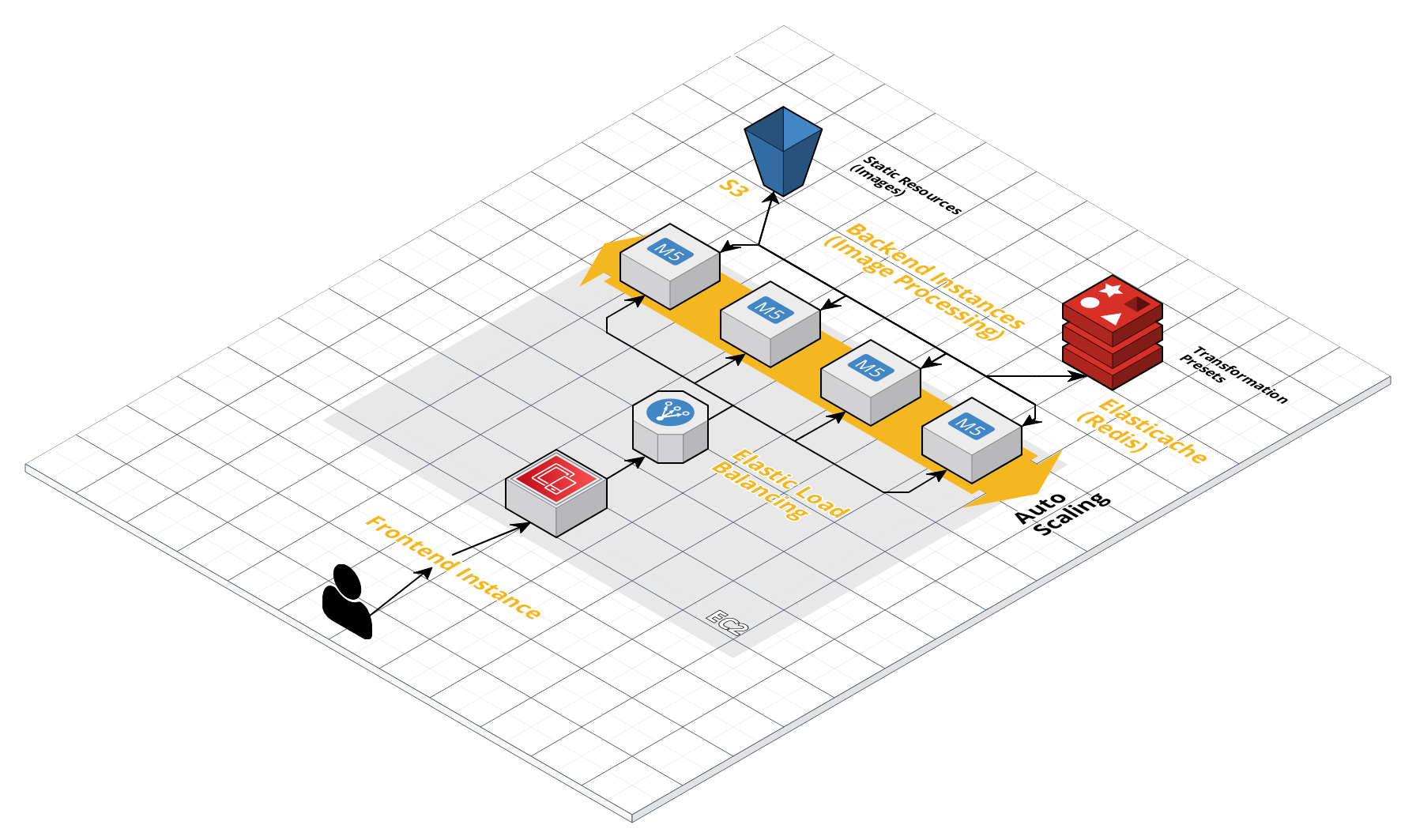
The images are displayed below the previously mentioned sections, the original shown on the left and the transformed image on the right. When a transformation is made, users have the option to either save the new image, which brings up a popup (shown below) that lets you choose a name for the file to either download or upload it again to the s3 bucket. Users can also choose to use it again as the input image, if further transformations need to be applied to the new image



#### Client / server demarcation of responsibilities

When a user interacts with the webpage, the frontend sends the data to the backend, where an elastic load balancer distributes the traffic across the auto scaling instances. More traffic means more instances are used to handle the load. Each instance interacts with S3 and Elasticache when necessary, retrieving image and preset data to be processed and sent back to the frontend to display the user’s desired outcome

#### Architecture Diagram



To further explain the flow of data, when the user interacts with the webpage, the requests are sent to the backend server. (Contained in docker alongside the frontend) Image transformation requests send the image and preset data to the Sharp API, that responds with the transformed image, which then the backend sends to the frontend to be displayed to the user

#### Context Diagram

Diagram

Description automatically generated

#### Response filtering / data object correlation

*Show us how you manipulated the data. The same comments apply about referring to the diagrams and supporting your work with code fragments as appropriate. We provide an example below of how to show this diagramatically. Please note that this example is quite specific to the system being explained. Yours might have an entirely different look, but do a similar job. This section will vary markedly according to the application and may not make much sense for some custom applications such as rendering.*

The way that data is manipulated is done in two parts. The preset data largely stays the same, being stored in the request body JSON (req.body) to be used for transformation. The function that calls sharp to make the transformations uses if statements to check if a transformation has been used, and applies if it has

The image data is stored in the s3 bucket as a Buffer object. Buffer is used as many file readers like the Sharp API can manipulate Buffer data, as they are suitable for fine-grained data manipulation, along with its ease of conversion to other data types like a base64 URL, used to re-render the images to the webpage so they can be displayed back to the user.

function transform(imgBuffer, transformation) {

// resize image

let transformationPipeline = sharp(imgBuffer).resize(transformation.width, transformation.height);

// greyscale

if (transformation.greyscale) {

transformationPipeline.greyscale();

}

// black and white

if (transformation.blackwhite) {

transformationPipeline.threshold(100);

}

// brightness

if (transformation.brightness) {

transformationPipeline.modulate({

brightness: transformation.brightness,

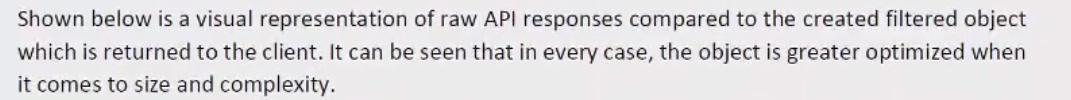
});

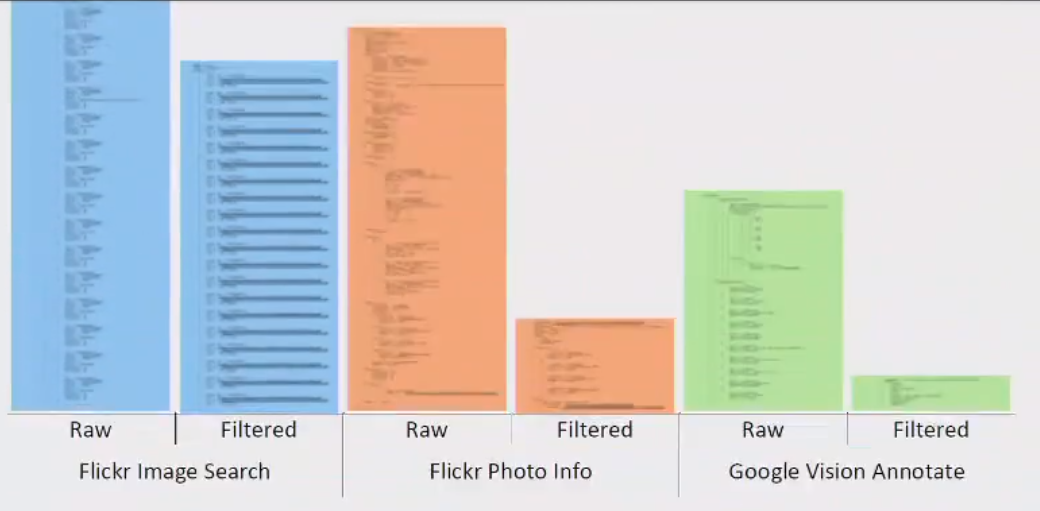
}

Buffer Array Example Base64 URL Example

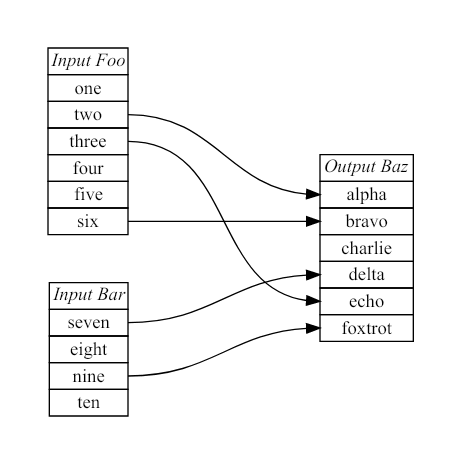
data:image/undefined;base64,iVBORw0KGgoAAAANSUhEUgAAAwwAAAHMCAIAAAD3arixAAAACXBIWXMAAAsTAAALEwEAmpwYAAAgAElEQVR4nO3dz2sjaX748fxLPjSmDz66N6w…

<Buffer 89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 03 0c 00 00 01 cc 08 02 00 00 00 f7 6a b8 b1 00 00 00 19 74 45 58 74 53 6f 66 74 77 61 72 65 00 ... 9986 more bytes>





*You could instead use a data relationship diagram:*



### Scaling and Performance

*This is a crucial aspect of the report, and you should use this section to document the approach taken to scaling – the nature of the application load, how it was varied and how the scaling infrastructure responded. You should refer to the architectural diagram above or reproduce the relevant aspects here. You should include screenshots of CPU, network or queuing metrics as observed on the cloud services dashboard, together with screenshots of your settings and the scaling pool instance creation and destruction. We expect that your work here will demonstrate successful scale out and scale in as required in the assignment specification. The screen shots that you use here will also very likely be re-purposed as part of the slide deck for the demo.*

*An example scaling image is shown below, and we would normally expect to see this sort of image and some evidence of your group settings. Note the instance count on the left hand axis. As noted, many alternatives are possible.*



### Test plan

*Manual testing is fine and our expectations are in line with the example grid below. You can show the results through a screen shot and point us to these from the table. Your tests should include:*

* *Positive outcome cases*
* *Negative outcome cases (error scenarios)*
* *Edge cases*
* *Non-functional cases*

*Note that the grid below is unrelated to this assignment.*



*As they are common in industry you could define your Acceptance Criteria as GWT statements. This is not compulsory, but see:* [*https://www.agilealliance.org/glossary/gwt/*](https://www.agilealliance.org/glossary/gwt/)*. And here is an example:*



Difficulties / Exclusions / unresolved & persistent errors /

*In this section, you should explain anything that caused you problems and how you overcame those problems. Tell us if there was any issue that prevented you completing the assignment to specification. Tell us about any assumptions or compromises that you have made. Those who worked with an API like Spotify, which presented particular concerns, should discuss the compromises here, and this is also where you can tell us about problems with API keys and responses.*

*More generally, you might consider:*

* *Your major roadblocks and how you resolved them.*
* *Any functionality you didn’t or couldn’t finish*
* *Are there any differences between your brief and what you delivered? If so, explain why.*
* *Are there any outstanding bugs?*

## Extensions (Optional)

*In this section, you can tell us if you wish to how you might extend your app and make it better. This is an opportunity to tell us about good ideas that you had that you didn’t have time to tell us about.*

## User guide

*Tell us how to use your application. You may re-use some of the screenshots from the use case descriptions, but this is more about how to use the app. As long as we can find what we need to do to use your application, this need not be all that long.*

*But either way, screenshots are your friend.*

## References

*Use a standard approach to referencing – see the guidance at* [*https://www.citewrite.qut.edu.au/cite/*](https://www.citewrite.qut.edu.au/cite/)*.*

Stack Abuse. (2020). Using Buffers in Node.js. Digital Ocean.

<https://www.digitalocean.com/community/tutorials/using-buffers-in-node-js>

## Appendices

*Stuff you want to include, but is too long or too complex to include in the main report text. The full Docker file, some longer excerpt from API docs. Whatever helps.*

*[Our thanks to those students who allowed us to use their work in the examples presented above.]*